

IceTeam 2



“Sandra Bullock”

Vice snapshot with CCS64 palette

**Made with the GIMP from a KM photo
and converted to C64 160x200
Multicolor Mode Bitmap
by Stefano Tognon
in 2002**

“Here we are again!”

...



Free Software Group

OSin 2
version 1.00
21 November 2002

General Index

Editorials.....	4
News.....	5
Sidplay2/w.....	5
JohnPlayer 1.6/2.0b.....	5
PSID64 0.5.....	5
TSID2 0.1.....	6
Ninjatrk 1.0/1.01/1.02.....	6
HVSC 5.1.....	7
Audio::SID 3.01 & SIDedit 3.01.....	7
The SID Compo II.....	7
Lasse Öörni (Cadaver) Interview!.....	8
David Wittaker's Lazy Jones music routine.....	10
The Code.....	10
Conclusion.....	20
Matt Gray's Driller music routine.....	21
Songs.....	21
Tracks and pattern.....	22
Notes.....	22
Instruments.....	23
Arpeggio.....	25
Vibrato.....	25
Special Effects.....	26
The Code.....	27
Conclusion.....	52

Hi, again.

In this new issue of SIDin I presents two reverse engineering works about two fantastic sid tunes: Lazy Jones and Driller. Even if you probably already know how the two authors made the sound in the tunes, maybe someone could be interesting too.

I hope that David and Matt (Matt! Where are you?) take this work for that is: a big recognition for their fantastic work in the C64 scene!

Remember that the right of the presented code remain to the two authors, so contact them if you want to get a business use of it.

See you at the next issue.

Bye

Some various news of players, programs and competition:

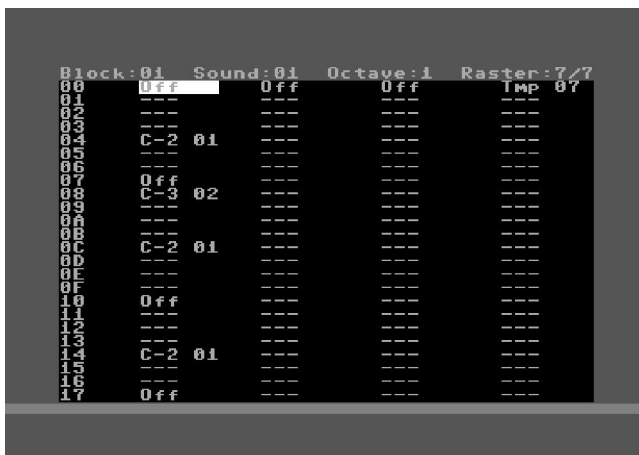
- Sidplay2/w
- JohnPlayer 1.6 and 2 beta
- PSID64 0.5
- TSID2 0.1
- Ninjatrak 1.0/1.01/1.02
- HVSC 5.1
- Audio::SID 3.01 & SIDedit 3.01
- The Sid Compo II

Sidplay2/w

Released on 20 Oct 2002 this new version of the Windows sid player supports the new RSID file format. RSID is just a PSID (with the new PSIDv2NG extension) that require a real C64 environment to run (Real SID). As a real C64 rip can crash old sid player, changing the PSID header in RSID allow that player to not find the sid as valid and so it did not crash.

Download from: <http://www.student.nada.kth.se/~d93-alo/c64/spw>

JohnPlayer 1.6/2.0b



The music player editor of Aleks Eeben is at version 1.6. This player uses only 7 rasterline, so it is very indicate for demo that need to use few rasterlines for the music.

It is also available a 2.0 beta version that has some advantage in the steps of sound table and sequencer, but now use 8 rasterlines.

Download the player from:
<http://www.student oulu.fi/~aeeben/>

PSID64 0.5

Released at 7 Nov 2002 this new version of the program that generate a C64 executable program from a sid file now support the RSID files.

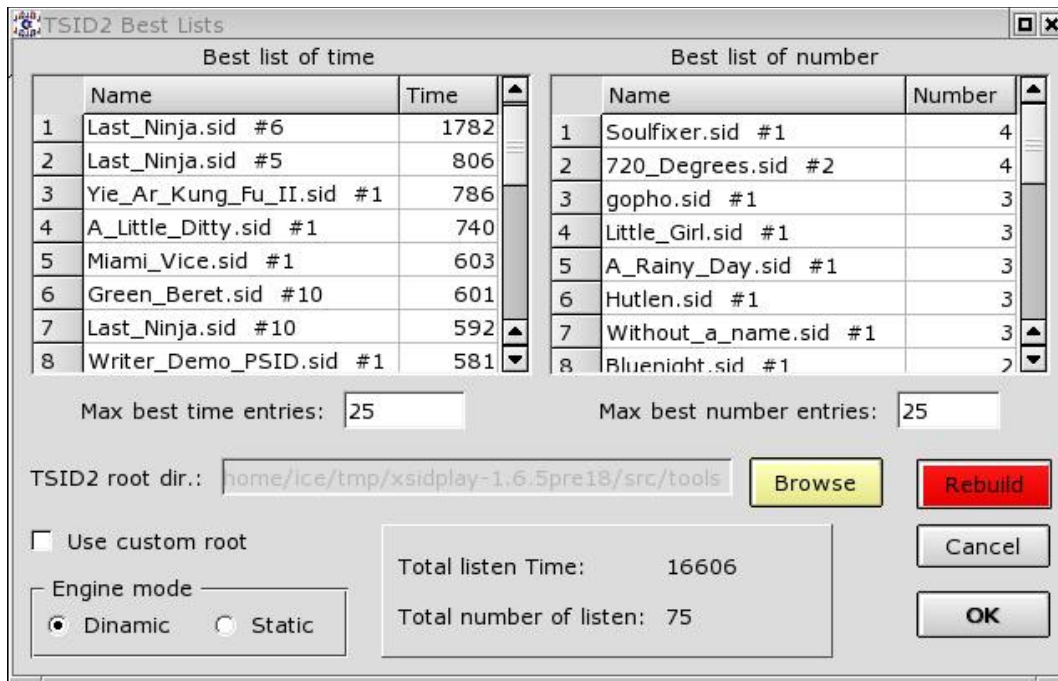
You can now listen to the real C64 rips on your C64 :-)

Download from <http://sf.net/projects/psid64>

TSID2 0.1

After some very long delay, finally a preliminary version of TSID2 (Time Sid Manager next generation) library is available on Linux system. The library store you listening statistic about sid tunes from a patched sid player, so you can look for your best sids.

It is totally based onto MD5 fingerprinter, so it can manages even reordered collection of sid tunes. You can use a patched xsidplay-1.6.5pre18 version that shows all the features of the new library.



Download all the stuff from <http://sf.net/projects/tsid>

Ninjatrk 1.0/1.01/1.02



Released on 1 Nov 2002 this is a new tracker by Lasse Öörni (Cadaver).

This tracker use only 10 rasterline and is optimized even for use very little quantity of memory.

It is a player that can be used for demo programs where memory and rasterlines are limited.

Version 1.01 was released 2 weeks later and version 1.02 some days after.

The last version has only 8 rasterlines even during badlines, so it is now more interesting.

Download from:

<http://covertbitops.c64.org/tools/ninjatrk.zip>

HVSC 5.1

Released on 26/10/2002 the upgrade 33 of HVSC contains the real rip (RSID).

After this update, the collection should contain 19,203 SID files!

This update contains:

- 504 new SIDs
- 49 fixed/better rips
- 90 fixes of PlaySID/Sidplay1 specific SIDs
- 7 repeats/bad rips eliminated
- 317 SID credit fixes

Download the collection/upgrade from: <http://www.hvsc.c64.org>

Audio::SID 3.01 & SIDedit 3.01

The LaLa perl module for managing sid file (Audio::SID) and the graphical program SIDedit is now at version 3.01.

SIDedit is a Perl/Tk GUI application that allows to read, write and edit binary SID files: the definitive utility for a SID ripper!

This version has some bug fixes and the support of RSID file format.

Download from the Lala site: <http://lala.c64.org>

The SID Compo II



The c64.sk Music Competition, from 8 October to 4 November 2002 has reached 30 entries from 13 counties and some other bonus tunes!

This year the competition was open to all the kind of music players, while the previous (Goattrk compo) was open only to Goattrk player.

Here the final classification:

PLACE	PTS	HANDLE / GROUP / SONG NAME / TIME
1	490	Dane / Crest trial_and_error
2	465	Vip/ Padua / Role / WOW / Cyber Zound Prancah
3	452	DaFunk / Onslaught Euro(S)cream
4	439	GH lafrique
	439	Yodelking of Defiers & UL-Tomten Banankorv
5	425	Dr.Voice / Magic_Carpet_Ride
6	420	Trance / X-Stayle Way_of_the_Saint
	420	No-XS / Toondichters Still_in_April
7	409	Cadaver / CovertBitops TheConsultant
8	395	Jammer / MSL / Samar Sky_Flight
9	388	Oedipus Wings_of_Death
10	382	Makt One / Fairlight NagChampa
	382	Sidder / MultiStyle Labs / Role MrPivo
11	379	Aleksi Eeben / CNCD RoundBlueRath
12	378	Smalltown Boy / MultiStyle Labs coffee
13	377	Luca / Fire mindfixer
14	375	Dalez / Creators / Rebels midnightrun
15	371	A Life in Hell / WOW / Unreal internalise
16	370	Mr.Death / Creators Russian_Cow_Moolok
17	368	Mermaid / Creators / Crest in_die_Dunkelheit
18	362	Richard / TND / Civitas / PTV TwiceDelight
19	348	St0fF / N30PLA51A Wektag
20	324	Waz / Padua Very_Doubtful_Space
21	322	TDS / Creators Level_Zero
22	317	hukka / Exec kauppa
23	313	CeRRor / Ex-Xentax CerroR_MoS
24	306	pOnToNiuS Poke
25	290	Pater PI/ Church64 Star_Fox_Corneria
26	279	Alias Medron / Padua / Samar relax
27	252	Six of DLoC /Dark Lords of Chaos Hi6i11y H34v3n!

See the even at: <http://www.c64.sk/index.php?content=article.php&articleid=57&id=832>

In this second issue, I present this interview with Lasse that occurs with some emails in November of this year. I like the Cadaver's works: fantastic games with a wonderful atmosphere like in MW series (e.g. I like the rain, or the day/night effect you can find in them) and a series of cross-develop tools and music players.

Hello, Lasse,

Before starting say something about your real life: where do you live, what do you do...

I live near Oulu, Finland, and study physics/math/chemistry/computer science in the Oulu university (I aim to be a teacher)

Well, you are probably known to the most by having created many big games for the C64: Escape From New York, BOFH:Servers Under Siege, Metal Warrior: you had wrote anything from engines, graphics, musics. Maybe you are one of the last full coder around that use however the modern technology (crossdevelop)?

This is an interesting question, because I started "serious" or C64 coding and crossdeveloping quite late, in 1998, after a break of several years; before that I never got anything noteworthy done on the C64. So I guess that makes me quite a newcomer. :)

But there are several coders using crossdevelopment, and more are picking it up, and big demos & games are being created, so perhaps I wouldn't like to say "one of the last".

How about your players? You had created Music driver, SadoTracker, GoatTrk and NinjaTrk. Why you had write so many players?

I've been steadily improving as a coder, so each time, when starting a new game project, I also wanted to rewrite the playroutine for memory or speed (rastertime) gain. Sometimes, I even wrote editors :)

I thought GoatTracker would be the ultimate for my personal use, because it was fast & easy to compose on, but finally I had to realize that music made with it takes too much memory (because of the patterns, they're like in Amiga/PC tracker programs). So I started the quest for the extremely optimized playroutine/musicdata combination (NinjaTracker) :)

Speaking about music: what we must expect from you now? Another player? Other features to one of your player (e.g. something had asked for digi-sample support in your player)?

I have no experience of sample coding (although I guess it's not too hard), so it's unlikely they'd be featured in my editor(s). Also, GoatTracker has kind of reached the peak in ease-of-use vs. features (it is missing step-programming for sounds, that would add versatility at the cost of easiness), and NinjaTracker in speed & size, so at least now I can't see the need/possibility for another player.

So, you can hope/expect that I make good music for MW4 :) (there are also other non-C64 related guys making music for it, I'll then arrange it for C64 as necessary -- same system as with MW3)

Now some quick final (standard) questions:

Real machine vs emulator: what do you think of?

Real machine is better for gaming sessions, that' s for sure. :)
And a real SID is very much preferable for composing.
But emulators are very powerful tools in testing your programs.

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

Worst is probably the music from Advanced Action Movie Simulator, it was done that way on purpose (crapgame-compo)

About best I don' t know, MW2 has some nice melodies in a couple of tunes.

Who are your best sid authors?

Quite the usual; Rob Hubbard, Martin Galway, Matt Gray, Jonathan Dunn (this guy composed the ultimate war-feeling/heroic SIDs), Reyn Ouwehand

What are the best sids ever in your opinion?

Hubbard' s Sanxion loader, and the various Ocean Loaders evoke the most feeling in me :)
Platoon, Last Ninja 2 & Last Ninja 3 soundtracks contain also nothing but brilliant tunes to me.
Usually, it' s not the technicality that I care most of, but rather the melodies and the atmosphere.

***Finally, many thanks for the time you give for this interview,
and now you can say any things you want for concluding this interview :)***

Ok, this goes for any C64 gamedevelopers. Beware of 2 things that will get your project cancelled, by mysterious hand of fate:

- 1) planning to do a Zelda project with bitmap scrolling
- 2) trying to do another Turricon for C64 :)

Webography:

Covert BitOps site:

<http://covertbitops.c64.org/>

Lasse Öörni homepage

<http://www.student oulu.fi/~loorni/>

David Wittaker' s Lazy Jones music routine

by Stefano Tognon <ice00@libero.it>



What to say about this game? Fantastic.

It was simple but very original and with a music that I think all who play it did not forget. Now I simply show you how David made this music. You probably will be surprise to see a so simple engine gives this magic music: maybe it is all David talent!

First of all: the reverse engineering work is based onto the Sid rip present in HVSC. How you see it is not complete with the sound SX: however it is just what we need.

If you look at the code you can see that the engine is very simple:

- The 20 music peaces are with the same length (the first is a case we see after): 80h bytes of data.
- Each peaces of music contains the low/high frequency to set in the sid registers for the 2 voices used (so 4 bytes).
- There' s not a note duration: each notes have the same length.
- There' snot special SID effect: all the notes use the same setting: Attack/Decay of 66h and 21h wave form.
- As the Decay/Released is not set (so 0), there' sa simple 20h to 21h setting in the wave form when the note changes.
- Tune 1 is an exception: it is the running of tune 10 and 11 in sequences.

So from where comes the beautiful sound? I think from the music notes themselves!

Maybe David had used some frequencies that give a better result using a little less or high values from the expected. However this require me many times for investigated; so look at the code and judge from yourself.

The Code

This code is a reverse engineering works, so contact the copyright owner author for making any business use of it.

I have also noted that the game was ripped from a cracked version of the game, as it contains a faulty track (not used in the rip, but present). You can see this track at the end of the listing at the *wrong* label, and you can listen to it if you change the source and you active it again.

```
processor 6502
org 1921

.byte "PSID"
.word $0200          ; version 2
.word $7C00          ; data offset
.byte #>init
.byte #<init
.byte #>init
.byte #<init
.byte #>IRQ
.byte #<IRQ
.word $1500          ; 21 song
.word $0100          ; default song
```

```

.word $FFFF
.word $FFFF
.byte "lazy Jones",0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
.byte "David Whittaker",0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
.byte "1984 Terminal Software",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

MUSIC = $03E0      ; 1=play music
INDEX = $03E1      ; index to freq. pattern
DELAY = $03E3      ; delay (speed of tune)
PATTL = $03E4      ; low pointer of pattern
PATTH = $03E5      ; high pointer of pattern

init:
    cmp    #$00
    beq    first
    tax
    lda    #$80
    sta    COMP+1
    lda    tableLo,x
    sta    LOP+1
    lda    tableHi,x
    sta    HIP+1
    jmp    initEngine

first:
    lda    #$00
    sta    COMP+1
    lda    #<Tune1
    sta    LOP+1
    lda    #>Tune1
    sta    HIP+1
    jmp    initEngine

tableLo:
    .byte <Tune1, <Tune2, <Tune3, <Tune4
    .byte <Tune5, <Tune6, <Tune7, <Tune8
    .byte <Tune9, <Tune10, <Tune11, <Tune12
    .byte <Tune13, <Tune14, <Tune15, <Tune16
    .byte <Tune17, <Tune18, <Tune19, <Tune20
    .byte <Tune21

tableHi:
    .byte >Tune1, >Tune2, >Tune3, >Tune4
    .byte >Tune5, >Tune6, >Tune7, >Tune8
    .byte >Tune9, >Tune10, >Tune11, >Tune12
    .byte >Tune13, >Tune14, >Tune15, >Tune16
    .byte >Tune17, >Tune18, >Tune19, >Tune20
    .byte >Tune21

IRQ:
    lda    MUSIC      ; read music flag
    beq    exit        ; 0=not play
    dec    DELAY       ; dec delay for speed
    bne    exit
    lda    #$0C
    sta    DELAY       ; init the delay (speed) of tune
    ldy    #$00
    lda    ($B4),y     ; read low frequency
    beq    skip1       ; zero?
    sta    $D400       ; Voice 1: Frequency control (lo byte)
    iny
    lda    ($B4),y     ; read high frequency
    sta    $D401       ; Voice 1: Frequency control (hi byte)
    lda    #$20
    sta    $D404       ; Voice 1: Control registers
    lda    #$21
    sta    $D404       ; Voice 1: Control registers

skip1:
    lda    $B4
    clc
    adc    #$02
    sta    $B4
    lda    $B5
    adc    #$00
    sta    $B5
    ldy    #$00
    lda    ($B4),y     ; read low frequency
    beq    skip2       ; zero?
    sta    $D407       ; Voice 2: Frequency control (lo byte)
    iny
    lda    ($B4),y     ; read high frequency
    sta    $D408       ; Voice 2: Frequency control (hi byte)
    lda    #$20
    sta    $D40B       ; Voice 2: Control registers
    lda    #$21
    sta    $D40B       ; Voice 2: Control registers

```

Tables with
to the tunes

```

skip2:
    lda    $B4
    clc
    adc    #$02
    sta    $B4
    lda    $B5
    adc    #$00
    sta    $B5
    inc    INDEX        ; inc index by 4
    inc    INDEX
    inc    INDEX
    inc    INDEX
    lda    INDEX

COMP:
    cmp    #$80        ; end of pattern?
    bne    exit
    lda    #$00
    sta    INDEX        ; set pointer to the begin
    lda    PATTL        ; repristinate pattern pointer
    sta    $B4
    lda    PATTH
    sta    $B5

exit:    jmp    $EA31        ; Default hardware interrupt (IRQ)

setAD:
    lda    #$1F
    sta    $D418        ; Select volume and filter mode
    lda    #$66
    sta    $D405        ; Generator 1: Attack/Decay
    sta    $D40C        ; Generator 2: Attack/Decay
    sei
    lda    #<IRQ
    sta    $0314        ; Vector: Hardware Interrupt (IRQ)
    lda    #>IRQ
    sta    $0315        ; Vector: Hardware Interrupt (IRQ)
    cli
    rts

initEngine:
    lda    #$01
    sta    MUSIC        ; play music
    lda    #$00
    sta    INDEX        ; reset index pointer
    lda    #$0C
    sta    DELAY        ; set delay (speed of tune)

LOP:
    lda    #$00
    sta    PATTL
    sta    $B4

HIP:
    lda    #$1C
    sta    PATTH
    sta    $B5
    lda    #$FF
    sta    $D416        ; Filter cut frequency: hi byte
    lda    #$03
    sta    $D417        ; Filter resonance control/voice input control
    jsr    setAD
    rts

org 2400
Tune2:
    .byte $D5, $07, $D2, $0F
    .byte $BE, $0F, $E6, $0F
    .byte $D5, $07, $FD, $07
    .byte $D2, $0F, $D1, $12
    .byte $D5, $07, $FD, $07
    .byte $D2, $0F, $C3, $11
    .byte $D5, $07, $FD, $07
    .byte $BE, $0F, $E6, $0F
    .byte $47, $06, $D1, $12
    .byte $7B, $0C, $A3, $0C
    .byte $33, $06, $5B, $06
    .byte $7B, $0C, $A3, $0C
    .byte $33, $06, $5B, $06
    .byte $33, $06, $5B, $06
    .byte $7B, $0C, $A3, $0C
    .byte $E9, $07, $D2, $0F
    .byte $BE, $0F, $E6, $0F
    .byte $D5, $07, $FD, $07
    .byte $D2, $0F, $D1, $12
    .byte $D5, $07, $FD, $07
    .byte $D2, $0F, $C3, $11
    .byte $D5, $07, $FD, $07
    .byte $BE, $0F, $E6, $0F
    .byte $68, $09, $DA, $0B
    .byte $BD, $12, $E5, $12

```

Now all the list of
frequencies to play

```

.byte $54, $09, $7C, $09
.byte $BD, $12, $E5, $12
.byte $54, $09, $7C, $09
.byte $BD, $12, $E5, $12
.byte $54, $09, $7C, $09
.byte $BD, $12, $E5, $12

```

Tune3:

```

.byte $E9, $07, $B5, $17
.byte $D2, $0F, $B5, $17
.byte $E9, $07, $1F, $15
.byte $D2, $0F, $D1, $12
.byte $D5, $07, $1F, $15
.byte $D2, $0F, $B5, $17
.byte $D5, $07, $FD, $07
.byte $BE, $0F, $E6, $0F
.byte $68, $09, $A2, $25
.byte $D1, $12, $31, $1C
.byte $68, $09, $B5, $17
.byte $D1, $12, $1F, $15
.byte $68, $09, $B5, $17
.byte $BD, $12, $1F, $15
.byte $54, $09, $7C, $09
.byte $BD, $12, $B5, $17
.byte $D9, $05, $F7, $05
.byte $C6, $0B, $EE, $0B
.byte $D9, $05, $F7, $05
.byte $C6, $0B, $EE, $0B
.byte $D9, $05, $F7, $05
.byte $C6, $0B, $EE, $0B
.byte $ED, $05, $B5, $17
.byte $DA, $0B, $C3, $11
.byte $D9, $05, $F7, $05
.byte $C6, $0B, $EE, $0B
.byte $D9, $05, $C3, $11
.byte $C6, $0B, $EE, $0B
.byte $D9, $05, $F7, $05
.byte $C6, $0B, $EE, $0B
.byte $D9, $05, $F7, $05
.byte $C6, $0B, $EE, $0B

```

Tune4:

```

.byte $0C, $07, $18, $0E
.byte $0E, $0E, $2C, $0E
.byte $0C, $07, $D2, $0F
.byte $18, $0E, $C3, $11
.byte $0C, $07, $C3, $11
.byte $18, $0E, $D1, $12
.byte $0C, $07, $C3, $11
.byte $18, $0E, $D2, $0F
.byte $ED, $05, $DA, $0B
.byte $C6, $0B, $EE, $0B
.byte $ED, $05, $18, $0E
.byte $C6, $0B, $EE, $0B
.byte $ED, $05, $D2, $0F
.byte $C6, $0B, $EE, $0B
.byte $E3, $05, $F7, $05
.byte $C6, $0B, $EE, $0B
.byte $0C, $07, $18, $0E
.byte $0E, $0E, $2C, $0E
.byte $0C, $07, $D2, $0F
.byte $18, $0E, $C3, $11
.byte $0C, $07, $C3, $11
.byte $18, $0E, $D1, $12
.byte $0C, $07, $C3, $11
.byte $18, $0E, $D2, $0F
.byte $ED, $05, $DA, $0B
.byte $C6, $0B, $EE, $0B
.byte $ED, $05, $18, $0E
.byte $C6, $0B, $EE, $0B
.byte $ED, $05, $B5, $17
.byte $DA, $0B, $D2, $0F
.byte $ED, $05, $B5, $17
.byte $DA, $0B, $A5, $1F

```

Tune5:

```

.byte $47, $06, $1E, $19
.byte $7B, $0C, $A3, $0C
.byte $3D, $06, $51, $06
.byte $8F, $0C, $31, $1C
.byte $3D, $06, $51, $06
.byte $7B, $0C, $A3, $0C
.byte $47, $06, $DF, $1D
.byte $7B, $0C, $A3, $0C
.byte $F7, $09, $1E, $19
.byte $DB, $13, $03, $14
.byte $E3, $09, $0B, $0A
.byte $EF, $13, $31, $1C
.byte $E3, $09, $0B, $0A
.byte $DB, $13, $03, $14

```

```

.byte $F7, $09, $DF, $1D
.byte $DB, $13, $03, $14
.byte $77, $07, $1E, $19
.byte $DB, $0E, $03, $0F
.byte $6D, $07, $81, $07
.byte $EF, $0E, $31, $1C
.byte $6D, $07, $81, $07
.byte $DB, $0E, $03, $0F
.byte $77, $07, $DF, $1D
.byte $DB, $0E, $03, $0F
.byte $68, $09, $1E, $19
.byte $BD, $12, $E5, $12
.byte $54, $09, $7C, $09
.byte $D1, $12, $31, $1C
.byte $54, $09, $7C, $09
.byte $BD, $12, $E5, $12
.byte $0C, $07, $DF, $1D
.byte $04, $0E, $2C, $0E

```

Tune6:

```

.byte $61, $08, $1F, $15
.byte $AF, $10, $D7, $10
.byte $4D, $08, $75, $08
.byte $C3, $10, $60, $16
.byte $4D, $08, $75, $08
.byte $AF, $10, $D7, $10
.byte $61, $08, $1F, $15
.byte $AF, $10, $D7, $10
.byte $47, $06, $D1, $12
.byte $7B, $0C, $A3, $0C
.byte $3D, $06, $51, $06
.byte $8F, $0C, $1F, $15
.byte $3D, $06, $51, $06
.byte $7B, $0C, $A3, $0C
.byte $47, $06, $D1, $12
.byte $7B, $0C, $A3, $0C
.byte $98, $05, $C3, $10
.byte $1C, $0B, $44, $0B
.byte $8E, $05, $A2, $05
.byte $30, $0B, $D1, $12
.byte $8E, $05, $A2, $05
.byte $1C, $0B, $44, $0B
.byte $98, $05, $C3, $10
.byte $1C, $0B, $44, $0B
.byte $47, $06, $D1, $12
.byte $7B, $0C, $A3, $0C
.byte $3D, $06, $51, $06
.byte $8F, $0C, $1E, $19
.byte $3D, $06, $51, $06
.byte $7B, $0C, $A3, $0C
.byte $47, $06, $A5, $1F
.byte $7B, $0C, $A3, $0C

```

Tune7:

```

.byte $30, $04, $D1, $12
.byte $4D, $08, $75, $08
.byte $30, $04, $1F, $15
.byte $61, $08, $C3, $10
.byte $26, $04, $3A, $04
.byte $4D, $08, $75, $08
.byte $30, $04, $1F, $15
.byte $4D, $08, $75, $08
.byte $B4, $04, $D1, $12
.byte $54, $09, $7C, $09
.byte $B4, $04, $C3, $10
.byte $68, $09, $18, $0E
.byte $AA, $04, $BE, $04
.byte $54, $09, $7C, $09
.byte $AA, $04, $BE, $04
.byte $54, $09, $7C, $09
.byte $98, $05, $60, $16
.byte $1C, $0B, $44, $0B
.byte $98, $05, $60, $16
.byte $30, $0B, $60, $16
.byte $8E, $05, $A2, $05
.byte $1C, $0B, $44, $0B
.byte $98, $05, $60, $16
.byte $1C, $0B, $44, $0B
.byte $47, $06, $60, $16
.byte $8F, $0C, $1F, $15
.byte $3D, $06, $51, $06
.byte $8F, $0C, $1F, $15
.byte $3D, $06, $51, $06
.byte $7B, $0C, $A3, $0C
.byte $3D, $06, $51, $06
.byte $7B, $0C, $A3, $0C

```

Tune8:

```

.byte $47, $06, $D2, $0F
.byte $7B, $0C, $A3, $0C

```

```

.byte $47, $06, $18, $0E
.byte $7B, $0C, $A3, $0C
.byte $30, $04, $C3, $10
.byte $4D, $08, $75, $08
.byte $30, $04, $D1, $12
.byte $61, $08, $1F, $15
.byte $47, $06, $D2, $0F
.byte $7B, $0C, $A3, $0C
.byte $47, $06, $18, $0E
.byte $7B, $0C, $A3, $0C
.byte $61, $08, $C3, $10
.byte $AF, $10, $D7, $10
.byte $4D, $08, $75, $08
.byte $AF, $10, $D7, $10
.byte $47, $06, $D2, $0F
.byte $7B, $0C, $A3, $0C
.byte $47, $06, $18, $0E
.byte $7B, $0C, $A3, $0C
.byte $30, $04, $C3, $10
.byte $4D, $08, $75, $08
.byte $30, $04, $D1, $12
.byte $61, $08, $1F, $15
.byte $0C, $07, $18, $0E
.byte $04, $0E, $2C, $0E
.byte $0C, $07, $C3, $10
.byte $18, $0E, $1F, $15
.byte $61, $08, $C3, $10
.byte $AF, $10, $D7, $10
.byte $61, $08, $1F, $15
.byte $C3, $10, $1E, $19

```

Tune9:

```

.byte $47, $05, $1E, $19
.byte $7B, $0A, $A3, $0A
.byte $47, $05, $1E, $19
.byte $8F, $0A, $B5, $17
.byte $47, $06, $1F, $15
.byte $7B, $0C, $A3, $0C
.byte $0C, $07, $1F, $15
.byte $18, $0E, $D1, $12
.byte $47, $05, $C3, $10
.byte $7B, $0A, $A3, $0A
.byte $47, $05, $C3, $10
.byte $8F, $0A, $D2, $0F
.byte $F4, $03, $18, $0E
.byte $D5, $07, $FD, $07
.byte $B4, $04, $18, $0E
.byte $68, $09, $DA, $0B
.byte $47, $05, $1E, $19
.byte $7B, $0A, $A3, $0A
.byte $47, $05, $1E, $19
.byte $8F, $0A, $A5, $1F
.byte $47, $06, $1F, $15
.byte $9F, $0C, $C7, $0C
.byte $0C, $07, $1F, $15
.byte $18, $0E, $1E, $19
.byte $E9, $07, $87, $21
.byte $BE, $0F, $E6, $0F
.byte $E9, $07, $87, $21
.byte $D2, $0F, $A5, $1F
.byte $0C, $07, $31, $1C
.byte $04, $0E, $2C, $0E
.byte $47, $06, $1E, $19
.byte $7B, $0C, $A3, $0C

```

Tune1:

Tune10:

```

; also tune 1 (with tune 11)
; . = low ^ = hi
; . H1 C2
; . H2 C3
; ^ H1 F#3
; ^ H2 D#3
; ^ H1 H2
; . H2 C3
; ^ H1 H-3
; . H2 C3
; ^ H1 D#3
; ^ H2 E3
; ^ H1 F3
; ^ H2 F#3
; ^ H1 H2
; . H2 C3
; . H1 C2
; . H2 C3
; ^ H1 D#3
; ^ H2 E3
; ^ H1 F3
; ^ H2 F#3
; ^ H1 E3
; . H2 C3

```

```

.byte $30, $04, $8F, $0A ; ^ H1 D#3
.byte $4D, $08, $75, $08 ; . H2 C3
.byte $30, $04, $61, $08 ; ^ H1 H2
.byte $4D, $08, $75, $08 ; . H2 C3
.byte $30, $04, $C3, $10 ; ^ H1 H-3
.byte $61, $08, $8F, $0C ; ^ H2 F#3
.byte $26, $04, $3A, $04 ; . H1 C2
.byte $4D, $08, $75, $08 ; . H2 C3
.byte $26, $04, $3A, $04 ; . H1 C2
.byte $4D, $08, $75, $08 ; . H2 C3

Tunel1:
.byte $30, $04, $8F, $0A ; ^ H1 D#3
.byte $4D, $08, $75, $08 ; . H2 C3
.byte $30, $04, $8F, $0A ; ^ H1 D#3
.byte $61, $08, $8F, $0A ; ^ H2 D#3
.byte $30, $04, $8F, $0A ; ^ H1 D#3
.byte $4D, $08, $75, $08 ; . H2 C3
.byte $30, $04, $30, $0B ; ^ H1 E3
.byte $61, $08, $68, $09 ; ^ H2 ...
.byte $3D, $06, $51, $06
.byte $7B, $0C, $A3, $0C
.byte $3D, $06, $51, $06
.byte $7B, $0C, $A3, $0C
.byte $3D, $06, $51, $06
.byte $7B, $0C, $A3, $0C
.byte $3D, $06, $51, $06
.byte $7B, $0C, $A3, $0C
.byte $47, $06, $68, $09
.byte $7B, $0C, $A3, $0C
.byte $47, $06, $68, $09
.byte $8F, $0C, $8F, $0A
.byte $47, $06, $68, $09
.byte $7B, $0C, $A3, $0C
.byte $47, $06, $61, $08
.byte $7B, $0C, $A3, $0C
.byte $30, $04, $61, $08
.byte $4D, $08, $75, $08
.byte $30, $04, $C3, $10
.byte $61, $08, $8F, $0C
.byte $26, $04, $3A, $04
.byte $4D, $08, $75, $08
.byte $26, $04, $3A, $04
.byte $4D, $08, $75, $08

Tunel2:
.byte $ED, $05, $C3, $11
.byte $C6, $0B, $EE, $0B
.byte $E3, $05, $F7, $05
.byte $DA, $0B, $EF, $13
.byte $E3, $05, $F7, $05
.byte $DA, $0B, $B5, $17
.byte $E3, $05, $F7, $05
.byte $DA, $0B, $EF, $13
.byte $70, $04, $4E, $0D
.byte $CD, $08, $F5, $08
.byte $66, $04, $7A, $04
.byte $E1, $08, $EF, $0E
.byte $66, $04, $7A, $04
.byte $E1, $08, $C3, $11
.byte $66, $04, $7A, $04
.byte $E1, $08, $EF, $0E
.byte $ED, $05, $C3, $11
.byte $C6, $0B, $EE, $0B
.byte $E3, $05, $F7, $05
.byte $DA, $0B, $EF, $13
.byte $E3, $05, $F7, $05
.byte $DA, $0B, $B5, $17
.byte $E3, $05, $F7, $05
.byte $DA, $0B, $EF, $13
.byte $E1, $08, $DF, $27
.byte $AF, $11, $D7, $11
.byte $CD, $08, $F5, $08
.byte $C3, $11, $86, $23
.byte $CD, $08, $F5, $08
.byte $C3, $11, $DF, $1D
.byte $CD, $08, $F5, $08
.byte $C3, $11, $9C, $1A

Tunel3:
.byte $68, $09, $D1, $12
.byte $BD, $12, $E5, $12
.byte $68, $09, $31, $1C
.byte $D1, $12, $60, $16
.byte $54, $09, $7C, $09
.byte $BD, $12, $E5, $12
.byte $68, $09, $D1, $12
.byte $BD, $12, $E5, $12
.byte $86, $03, $C3, $10
.byte $02, $07, $16, $07

```



```

.byte $86, $03, $31, $1C
.byte $0C, $07, $1F, $15
.byte $7C, $03, $90, $03
.byte $02, $07, $16, $07
.byte $86, $03, $C3, $10
.byte $02, $07, $16, $07
.byte $47, $06, $D2, $0F
.byte $7B, $0C, $A3, $0C
.byte $47, $06, $1E, $19
.byte $8F, $0C, $D1, $12
.byte $3D, $06, $51, $06
.byte $7B, $0C, $A3, $0C
.byte $3D, $06, $51, $06
.byte $7B, $0C, $A3, $0C
.byte $23, $03, $D2, $0F
.byte $3D, $06, $51, $06
.byte $23, $03, $D1, $12
.byte $47, $06, $D2, $0F
.byte $23, $03, $A2, $25
.byte $47, $06, $87, $21
.byte $23, $03, $A5, $1F
.byte $3D, $06, $51, $06

```

Tune14:

```

.byte $30, $04, $87, $21
.byte $61, $08, $87, $21
.byte $30, $04, $87, $21
.byte $61, $08, $87, $21
.byte $30, $04, $87, $21
.byte $61, $08, $87, $21
.byte $30, $04, $87, $21
.byte $61, $08, $87, $21
.byte $30, $04, $87, $21
.byte $61, $08, $3C, $32
.byte $30, $04, $87, $21
.byte $61, $08, $3C, $32
.byte $30, $04, $87, $21
.byte $61, $08, $3C, $32
.byte $47, $06, $3C, $32
.byte $7B, $0C, $A3, $0C
.byte $3D, $06, $51, $06
.byte $7B, $0C, $A3, $0C
.byte $3D, $06, $51, $06
.byte $7B, $0C, $A3, $0C
.byte $3D, $06, $51, $06
.byte $7B, $0C, $A3, $0C
.byte $3D, $06, $51, $06
.byte $7B, $0C, $A3, $0C
.byte $47, $06, $3C, $32
.byte $7B, $0C, $A3, $0C
.byte $47, $06, $3E, $2A
.byte $7B, $0C, $A3, $0C

```

Tune15:

```

.byte $FB, $04, $EF, $13
.byte $E3, $09, $0B, $0A
.byte $F1, $04, $05, $05
.byte $F7, $09, $EF, $0E
.byte $F1, $04, $05, $05
.byte $E3, $09, $0B, $0A
.byte $FB, $04, $EF, $13
.byte $E3, $09, $0B, $0A
.byte $A7, $06, $EF, $13
.byte $E3, $09, $0B, $0A
.byte $9D, $06, $B1, $06
.byte $F7, $09, $C3, $10
.byte $9D, $06, $B1, $06
.byte $E3, $09, $0B, $0A
.byte $A7, $06, $EF, $13
.byte $E3, $09, $0B, $0A
.byte $77, $07, $EF, $13
.byte $DB, $0E, $F8, $0E
.byte $6D, $07, $81, $07
.byte $EF, $0E, $D1, $12
.byte $6D, $07, $81, $07
.byte $DB, $0E, $03, $0F
.byte $77, $07, $C3, $10
.byte $DB, $0E, $03, $0F
.byte $EF, $0E, $D1, $12
.byte $CB, $1D, $F3, $1D
.byte $DB, $0E, $03, $0F
.byte $DF, $1D, $EF, $13
.byte $DB, $0E, $03, $0F
.byte $CB, $1D, $F3, $1D
.byte $EF, $0E, $60, $16
.byte $CB, $1D, $F3, $1D

```

Tune16:

.byte	\$30,	\$04,	\$A2,	\$25
.byte	\$4D,	\$08,	\$75,	\$08
.byte	\$30,	\$04,	\$3E,	\$2A
.byte	\$4D,	\$08,	\$75,	\$08
.byte	\$30,	\$04,	\$87,	\$21
.byte	\$4D,	\$08,	\$75,	\$08
.byte	\$30,	\$04,	\$C3,	\$10
.byte	\$61,	\$08,	\$1E,	\$19
.byte	\$26,	\$04,	\$3A,	\$04
.byte	\$4D,	\$08,	\$75,	\$08
.byte	\$26,	\$04,	\$3A,	\$04
.byte	\$4D,	\$08,	\$75,	\$08
.byte	\$26,	\$04,	\$3A,	\$04
.byte	\$4D,	\$08,	\$75,	\$08
.byte	\$30,	\$04,	\$1E,	\$19
.byte	\$4D,	\$08,	\$75,	\$08
.byte	\$30,	\$04,	\$C3,	\$10
.byte	\$4D,	\$08,	\$75,	\$08
.byte	\$30,	\$04,	\$87,	\$21
.byte	\$4D,	\$08,	\$75,	\$08
.byte	\$30,	\$04,	\$3E,	\$2A
.byte	\$61,	\$08,	\$A2,	\$25
.byte	\$3D,	\$06,	\$51,	\$06
.byte	\$7B,	\$0C,	\$A3,	\$0C
.byte	\$3D,	\$06,	\$51,	\$06
.byte	\$7B,	\$0C,	\$A3,	\$0C
.byte	\$3D,	\$06,	\$51,	\$06
.byte	\$7B,	\$0C,	\$A3,	\$0C
.byte	\$3D,	\$06,	\$51,	\$06
.byte	\$7B,	\$0C,	\$A3,	\$0C

Tune17:

.byte	\$26,	\$04,	\$3A,	\$04
.byte	\$7B,	\$0C,	\$A3,	\$0C
.byte	\$00,	\$00,	\$00,	\$00
.byte	\$1C,	\$0B,	\$44,	\$0B
.byte	\$30,	\$04,	\$8F,	\$0C
.byte	\$E3,	\$09,	\$0B,	\$0A
.byte	\$54,	\$09,	\$7C,	\$09
.byte	\$4D,	\$08,	\$75,	\$08
.byte	\$47,	\$06,	\$E9,	\$07
.byte	\$4D,	\$08,	\$75,	\$08
.byte	\$00,	\$00,	\$00,	\$00
.byte	\$54,	\$09,	\$7C,	\$09
.byte	\$47,	\$06,	\$61,	\$08
.byte	\$C1,	\$07,	\$E9,	\$07
.byte	\$9D,	\$06,	\$B1,	\$06
.byte	\$3D,	\$06,	\$51,	\$06
.byte	\$FB,	\$04,	\$98,	\$05
.byte	\$3D,	\$06,	\$51,	\$06
.byte	\$00,	\$00,	\$00,	\$00
.byte	\$9D,	\$06,	\$B1,	\$06
.byte	\$FB,	\$04,	\$47,	\$06
.byte	\$8E,	\$05,	\$A2,	\$05
.byte	\$F1,	\$04,	\$05,	\$05
.byte	\$8E,	\$05,	\$A2,	\$05
.byte	\$47,	\$06,	\$0C,	\$07
.byte	\$D5,	\$07,	\$FD,	\$07
.byte	\$00,	\$00,	\$00,	\$00
.byte	\$4D,	\$08,	\$75,	\$08
.byte	\$47,	\$06,	\$E9,	\$07
.byte	\$9D,	\$06,	\$B1,	\$06
.byte	\$F1,	\$04,	\$05,	\$05
.byte	\$AA,	\$04,	\$BE,	\$04

Tune18:

.byte	\$ED,	\$05,	\$B5,	\$17
.byte	\$C6,	\$0B,	\$EE,	\$0B
.byte	\$ED,	\$05,	\$1F,	\$15
.byte	\$DA,	\$0B,	\$B5,	\$17
.byte	\$E3,	\$05,	\$F7,	\$05
.byte	\$DA,	\$0B,	\$1F,	\$15
.byte	\$ED,	\$05,	\$B5,	\$17
.byte	\$DA,	\$0B,	\$A5,	\$1F
.byte	\$ED,	\$05,	\$B5,	\$17
.byte	\$C6,	\$0B,	\$EE,	\$0B
.byte	\$ED,	\$05,	\$1F,	\$15
.byte	\$DA,	\$0B,	\$B5,	\$17
.byte	\$E3,	\$05,	\$F7,	\$05
.byte	\$DA,	\$0B,	\$1F,	\$15
.byte	\$ED,	\$05,	\$B5,	\$17
.byte	\$DA,	\$0B,	\$A5,	\$1F
.byte	\$D5,	\$07,	\$FD,	\$07
.byte	\$BE,	\$0F,	\$E6,	\$0F
.byte	\$E9,	\$07,	\$C3,	\$10
.byte	\$D2,	\$0F,	\$D1,	\$12

```

.byte $E9, $07, $C3, $10
.byte $BE, $0F, $E6, $0F
.byte $D5, $07, $FD, $07
.byte $D2, $0F, $18, $0E
.byte $D5, $07, $D2, $0F
.byte $BE, $0F, $E6, $0F
.byte $D5, $07, $FD, $07
.byte $BE, $0F, $E6, $0F
.byte $D5, $07, $FD, $07
.byte $BE, $0F, $E6, $0F
.byte $D5, $07, $FD, $07
.byte $BE, $0F, $E6, $0F

```

Tune19:

```

.byte $30, $04, $EF, $0E
.byte $61, $08, $18, $0E
.byte $30, $04, $8F, $0C
.byte $61, $08, $8F, $0A
.byte $26, $04, $3A, $04
.byte $61, $08, $F7, $09
.byte $30, $04, $8F, $0A
.byte $4D, $08, $75, $08
.byte $30, $04, $C3, $10
.byte $61, $08, $EF, $0E
.byte $30, $04, $8F, $0C
.byte $61, $08, $8F, $0A
.byte $26, $04, $3A, $04
.byte $4D, $08, $75, $08
.byte $26, $04, $3A, $04
.byte $4D, $08, $75, $08
.byte $98, $05, $C3, $10
.byte $30, $0B, $EF, $0E
.byte $98, $05, $18, $0E
.byte $30, $0B, $8F, $0C
.byte $8E, $05, $A2, $05
.byte $1C, $0B, $44, $0B
.byte $98, $05, $8F, $0C
.byte $1C, $0B, $44, $0B
.byte $47, $06, $EF, $0E
.byte $8F, $0C, $18, $0E
.byte $47, $06, $8F, $0C
.byte $8F, $0C, $8F, $0A
.byte $3D, $06, $51, $06
.byte $8F, $0C, $F7, $09
.byte $47, $06, $8F, $0A
.byte $7B, $0C, $A3, $0C

```

Tune20:

```

.byte $AA, $04, $BE, $04
.byte $54, $09, $7C, $09
.byte $B4, $04, $D1, $12
.byte $54, $09, $7C, $09
.byte $B4, $04, $D1, $12
.byte $54, $09, $7C, $09
.byte $B4, $04, $D1, $12
.byte $68, $09, $C3, $10
.byte $B4, $04, $1F, $15
.byte $54, $09, $7C, $09
.byte $B4, $04, $D1, $12
.byte $68, $09, $18, $0E
.byte $AA, $04, $BE, $04
.byte $68, $09, $D1, $12
.byte $AA, $04, $BE, $04
.byte $54, $09, $7C, $09
.byte $8E, $05, $A2, $05
.byte $1C, $0B, $44, $0B
.byte $98, $05, $C3, $10
.byte $1C, $0B, $44, $0B
.byte $98, $05, $C3, $10
.byte $1C, $0B, $44, $0B
.byte $98, $05, $C3, $10
.byte $30, $0B, $D1, $12
.byte $0C, $07, $C3, $10
.byte $04, $0E, $2C, $0E
.byte $0C, $07, $18, $0E
.byte $04, $0E, $2C, $0E
.byte $02, $07, $16, $07
.byte $04, $0E, $2C, $0E
.byte $02, $07, $16, $07
.byte $04, $0E, $2C, $0E

```

Tune21:

```

.byte $1C, $04, $44, $04
.byte $61, $08, $C3, $10
.byte $30, $04, $EF, $13
.byte $61, $08, $60, $16
.byte $30, $04, $1E, $19
.byte $4D, $08, $75, $08
.byte $1C, $04, $44, $04
.byte $4D, $08, $75, $08

```

```

.byte $1C, $04, $44, $04
.byte $61, $08, $C3, $10
.byte $30, $04, $EF, $13
.byte $61, $08, $60, $16
.byte $30, $04, $1E, $19
.byte $61, $08, $9C, $1A
.byte $30, $04, $1E, $19
.byte $61, $08, $EF, $13
.byte $98, $05, $60, $16
.byte $1C, $0B, $44, $0B
.byte $84, $05, $AC, $05
.byte $1C, $0B, $44, $0B
.byte $47, $06, $EF, $13
.byte $7B, $0C, $A3, $0C
.byte $47, $06, $1E, $19
.byte $7B, $0C, $C3, $10
.byte $26, $04, $3A, $04
.byte $4D, $08, $75, $08
.byte $1C, $04, $44, $04
.byte $4D, $08, $75, $08
.byte $1C, $04, $44, $04
.byte $4D, $08, $75, $08
.byte $1C, $04, $44, $04
.byte $4D, $08, $75, $08

```

wrong:

```

.byte $3F, $3F, $37, $7F
.byte $6F, $7F, $6F, $1F
.byte $FC, $FC, $EC, $FE
.byte $F6, $FE, $F6, $F8
.byte $E0, $FF, $EF, $FF
.byte $EF, $EF, $FF, $E0
.byte $07, $FF, $F7, $FF
.byte $F7, $F7, $FF, $07
.byte $00, $00, $18, $3C
.byte $3C, $3C, $18, $00
.byte $3E, $63, $C6, $C0
.byte $E0, $F1, $7F, $3E
.byte $00, $00, $3C, $66
.byte $C6, $C7, $6C, $38
.byte $00, $00, $3C, $66
.byte $CC, $C1, $66, $3C
.byte $30, $30, $32, $6D
.byte $78, $F0, $D9, $CE
.byte $60, $60, $61, $F3
.byte $C6, $C6, $CE, $7B
.byte $01, $03, $B0, $F3
.byte $63, $66, $6E, $B7
.byte $98, $18, $19, $33
.byte $31, $70, $B6, $1B
.byte $00, $00, $F0, $18
.byte $80, $60, $33, $E3
.byte $00, $00, $00, $00
.byte $FF, $00, $00, $00
.byte $18, $18, $18, $18
.byte $18, $18, $18, $18
.byte $00, $00, $00, $00
.byte $0F, $1C, $18, $18

```

Conclusion

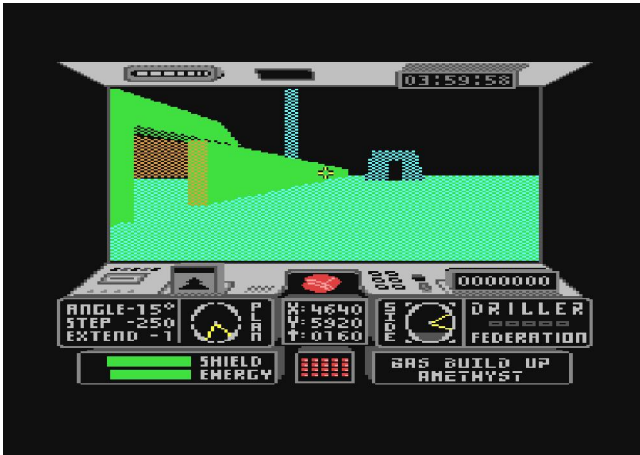
Well, as you see this is a very simple engine music (I can say that it is the most simple I had seen until now). However don' forget that the game is from 1984, so it was develop at the beginning of the Commodore time.

The last thing you may want to know is that this engine use a maximum of 4 rasterlines when it is piking a new note, and only 1 rasterline in the other case.

Matt Gray' s Driller music routine

by Stefano Tognon <ice00@libero.it>

Driller is one of the tune that I listen more often in the past. I built a special cable for recording the tune in the stereo from the C64, for listen it every day.



Else this tune is that make me found the HVSC in the net in late ' 99 when I search for a Driller image file, and I so found that I' mnot the unique people that was listen to sid music.

Th code presented is a my reverse engineering work about the psid rip of Driller that is present in the HVSC. So this is probably not a 100% version of the engine, as the rip did not contains the sound fx. However in the code you can see a not used notes pattern.

I have not yet investigate how the Matt' sen-
gine is evolved during the time, however Matt had used other music engine like RockMonitor 2/3 and MusicMaster.

Songs

The engine is based onto songs. There is a table (*songs*) shows below where there are to insert the pointers to the songs tracks.

```
songs =*
ltr1:
    .byte $00
    .byte <drillerMainTr1

htr1:
    .byte $00
    .byte >drillerMainTr1

ltr2:
    .byte $00
    .byte <drillerMainTr2

htr2:
    .byte $00
    .byte >drillerMainTr2

ltr3:
    .byte $00
    .byte <drillerMainTr3

htr3:
    .byte $00
    .byte >drillerMainTr3
```

Song 0 is for setting the sound to zero volume, where other number set the tracks for that song (in Driller song 1 is the main theme). You have to add the pointers (low/high) if you want to have more songs used by the engine.

For playing one song is simple: set his number in the *songsNumber* variable. The engine will start the selected song and the variable will be set to \$AB (that means current song).

Tracks and pattern

Each songs is made by 3 tracks and each tracks contains the number of the pattern to play. For examples:

```
drillerMainTr1:
    .byte $01, $01, $07, $09
    .byte $09, $09, $01, $07
```

In the first track of the song 1, it is to play in sequence the patterns 01, 01, 07, 09 ... and so on. Two special pattern numbers are reserved: \$FF restart the execution of the track, while the \$FE stop the sound when it is reached.

Each pattern is a pointer to data contains notes (as we will see in next paragraph), so this pointer are to be declared in two tables: *patptl* and *patpth*:

```
;low pointers
patptl =*
    .byte <pnt00
    .byte <pnt01
    .byte <pnt02
    ...

;high pointers
patpth =*
    .byte >pnt00
    .byte >pnt01
    .byte >pnt02
    ...
```

Notes

Each patterns contain the sequences of notes to play.

<i>byte</i>	<i>description</i>
\$00	Made gate = 0 for the previous note
\$xx	The xx number of note to play
\$FA nn	Select instrument number <i>nn</i>
\$FB mm	Make a negative portamento of value <i>mm</i>
\$FC pp	Make a positive portamento of value <i>pp</i>
\$FD kk	Set the duration of notes to <i>kk</i>
\$FF	End of notes in this pattern

Let I make some examples of pattern and data:

```
pnt27 =*
    .byte $FA, $0C
    .byte $FD, $01
    .byte $31, $3D, $49, $3D
    .byte $31, $3D, $49, $3D
    .byte $FF
```

In this pattern, it is selected to use the instrument \$0C, then to use a very short sound duration

(\$01) and so to play notes number \$31, \$3D, \$49, \$3D

```
pnt18 =*  
  .byte $FA, $13  
  .byte $FD, $07  
  .byte $FC, $37, $45  
  .byte $FD, $2F  
  .byte $47  
  .byte $FD, $07  
  .byte $FB, $7F, $47  
  ...
```

In this pattern it is select instrument number \$13, length of \$07, a positive portamento on note \$45 with intensity \$37. Then note \$47 is played for \$2F time and then, for \$07 time a negative portamento of intensity \$7F is apply to note \$47.

A special case is note \$00. When this note is reached, the previous note is played with the same settings, but with the gate bit set to 0, so the release phase will start. You can heart this effect in the last long sound at the end of the tune.

Instruments

The instruments contains many timbre effect like arpeggio, pulse wave routine and waveforms switch routine (useful for drum sound). Else it contains portamento and vibrato effects that can be apply to the instruments.

The instruments are specified by compiling two tables of 8 bytes each.

In the first there are to insert these values:

Byte	Description
Hi/Lo pulse of Wave waveform	If you use a rectangular waveform, this byte specify the pulse it may use. As the pulse is set by two sid registers, here there is only two nibble used as in this examples: value: xxyy (xx and yy = nibble) low of pulsed set: xx00 high of pulsed set: 00yy
Control register	This is the value used for setting the sid control register for this instrument
Attack/Decay	The AD sid value for the instrument
Sustain/Release	The SR sid value for the instrument

Byte	Description
Wave amplitude inc/dec value	<p>This value is for rectangular waveform and control how much increment/decrement must be used for a pulse variable effect (you can think it as the speed step of variation of the duty cycle). The direction of the variation is changed when reaching the max and min value for the pulse width.</p> <p>These value are hardcore in the engine and are 08 for the min and 0E for the max (so the high byte of pulse goes from \$08 to \$0E - pulse from 2048 to 3584-).</p>
hi/lo value for arpeggio effect	<p>This control the arpeggio effect to apply to the note.</p> <p>The value xxyy is to be see as:</p> <p>xx: the number of notes to add in sequence for the arpeggio</p> <p>yy: the pointer index to a table that contains the values to add to the note</p> <p>See the example for how the table can be compiled</p>
Control register	Control register set at the end of the note, but in the same frame of when setting the new note.
Instrument effects	<p>This is a bits field value that control witch effects to apply:</p> <p>bit 0: (1) a frequency/wave effect</p> <p>bit 1: (2) re-set instrument pulse at each new note</p> <p>bit 3: (4) waveform switch</p>

The second table contains the other values that can be set for the instruments:

Byte	Description
vibrato freq. value	Value of the vibrato frequency used for add/sub
vibrato length	Length of the vibrato effect (see example)
Control register	Control used for the first two frames when effect of bit 2 is on, and even when effect of bit 1 is on
portamento value	Value of the portamento: it is added/subtracted from the LOW of HIGH byte of frequency according to the portamento flag
portamento flag	<p>This byte specify the type of instrument portamento to achieve:</p> <p>\$00: no portamento for the instrument</p> <p>\$01: negative portamento LOW</p> <p>\$02: positive portamento LOW</p> <p>\$03. negative portamento HIGH</p> <p>\$xx: positive portamento HIGH</p>

Byte	Description
length	duration cycle for effect 1
0	Not used
0	Not used

Arpeggio

Now I illustrate how to use the arpeggio table.
First, it can be create a table like this:

```

PointerLo:
    .byte    <Table0
PointerHi:
    .byte    >Table0
    .byte    <Table1
    .byte    >Table1
    .byte    <Table2
    .byte    >Table2

```

Then it must be create the sequence of values (halfnote) to add for the arpeggio:

```

Table0:
    .byte    $00, $08, $12    ; length to use is 3
Table1:
    .byte    $00, $08        ; length to use is 2
Table2:
    .byte    $00, $04, $08    ; length to use is 3

```

So, a value of 30 in the arpeggio effect means: use arpeggio' sTable0 and 3 values, while a value of 22 means: use arpeggio' s Table2 and 2 values (instead of the 3 inserted in the table).

As you can see this method allow to use a very complex arpeggio effect in the tune

Vibrato

The vibrato routine is quite complex as it is very configurable. You can set the frequency value and the length (intensity) of the vibrato (the byte 0 and 1 in the instrument table 2).

Let me call V the first value and L the second one and F the actual frequency. The vibrato routine made so this cycle:

0	1	2	3	4
$F=F-V$	$F=F+V$	$F=F+V$	$F=F-V$	$F=F-V$
(L times)	(L times)	(L times)	(L times)	(L times)
$F=F-V$	$F=F+V$	$F=F+V$	$F=F-V$	$F=F-V$

The cycle start from 0, but when reached 4 it restart from position 1.

So as examples a value of V=10 and L=2 with an A4 note (440Hz) will produce this sequence of frequency:

430, 420, 430, 440, 450, 460, 450, 440, 430, 420, 430, ecc.

As we see in the first instrument' table, there are some effects that are activated by the last byte that are associated with the instrument:

- bit 0: This bit activated a frequency/wave effect that use byte 5 of second table as parameter.
- bit 1: Use the wave pulse set in the instrument at each note that are to be played. If this bit is not set, the last pulsed that was used in the last note is replicated for the new note.
- bit 2: Waveform switch: when this is set in the first two frames it is used the waveform that comes for the control register in the byte 3 of the second instrument table. Then the normal waveform is used.

So, bit 1 is to be used for maintaining the timbre of the instrument for each note or for not having a difference in the sound from one note to another. Bit 2 is to be used for drums effect.

Bit 0 is for making a effect that I don' know how to call, but I can describe: essentially the sound at a given frequency is played with control register from byte 2 of second table and after with a noise sound that is played with a high frequency that is constantly decrease and putted out after an xor-operation to 23h constant. This sequence continue and the length of the decrease seems given by the 5° byte of the second table (but the code seems to use a different comportment) .

So, an example: if control byte is C, and the cycle is of 1, with high frequency equals to F we have at each clock tick (note that if there' sportamento for the instrument, the F used is the actual value it were reached):

```
D404=C      D401=F
D401=81h    D401=(F-1) xor 23h
and so one.
```

If now the cycle is of 3, we have:

```
D404=C      D401=F
D401=81h    D401=(F-1) xor 23h
D404=C      D401=F
D401=81h    D401=(F-1) xor 23h
D404=C      D401=F
D401=81h    D401=(F-1) xor 23h
```

Note that at the first tick, before setting D404=C the player set D404=control register of instrument at byte 1 of first table.

Even if from this description seems that the length gives not a different result, it is possible that my analysis is faulty (I have not yet experimented this flag)

The Code

This code is a reverse engineering works, so contact the copyright owner author for making any business use of it.

```
; Driller by Matt Gray 1997
; extracted from the sid in HVSC
; NOTE: the prg goes at VIC speed as I use this for misuring rastertime:
; activate the CIA for having the true speed.
```

[illegible]

```

    jmp    decCycleTimer

makeCurrent:
    ldy    instrInd,x                ; index of instrument data
    lda    instr1+7,y                ; load instrument effect
    and    #$04                      ; is effect 4?
    beq    noEffect4

; change the control register for making a sound effect
controlEffect:
effect4:
    lda    dur4Eff,x                ; read duration of effect 4 switch
    beq    applyNormalControl

    dec    dur4Eff,x                ; dec duration of effect 4 switch
    lda    instr2+2,y                ; control register
    sta    $D404,x                  ; Voice 1: Control registers
    bne    noEffect4

applyNormalControl:
    lda    instr1+1,y                ; control register
    sta    $D404,x                  ; Voice 1: Control registers

noEffect4:
    lda    actSpeed                  ; actual cycle timer (speed of song)
    bne    noDelayEnd
    dec    actNoteLength,x           ; actual note length duration
    bmi    readPattern               ; jump if note is finished
noDelayEnd:
    jmp    pulseTimbre

; init tracks with selected songs

initTr =*
    ldy    songsNumber               ; load song number (1, 2)
    lda    ltr1,y                    ; set current track 1
    sta    currentTr1+0              ; current track 1 position (base)
    lda    htr1,y                    ;
    sta    currentTr1+1              ; current track 1 position (base)

    lda    ltr2,y                    ; set current track 2
    sta    currentTr2+0              ; current track 2 position (base)
    lda    htr2,y                    ;
    sta    currentTr2+1              ; current track 2 position (base)

    lda    ltr3,y                    ; set current track 3
    sta    currentTr3+0              ; current track 3 position (base)
    lda    htr3,y                    ;
    sta    currentTr3+1              ; current track 3 position (base)

    lda    songSpeed,y               ; read the song speed
    sta    speed                     ; cycle timer (speed of song)
    jmp    initMusic

; decrement the cycle timer (speed of song)
decCycleTimer =*
    cpx    #$0E                     ; is voice 3 ?
    bne    skipDec                   ; no, jump
    dec    actSpeed                  ; actual cycle timer (speed of song)
    bpl    skipDec                   ; jump if positive
    lda    speed                     ; cycle timer (speed of song)
    sta    actSpeed                  ; actual cycle timer (speed of song)
skipDec:
    lda    #$AB                      ; current song
    sta    songsNumber
    rts

; give in FD-FE the pattern pointer to a note
readPattern:
    lda    currentTr1+0,x            ; current track 1 position (base)
    sta    $FB                       ;
    lda    currentTr1+1,x            ; current track 1 position (base)
    sta    $FC                       ; FB-FC: track pattern pointer

    ldy    offsetTr1,x               ; read actual track position (offset)
    lda    ($FB),y                   ; read actual track pattern pointer value
    tay

    lda    patptl,y                  ; read pattern pointer low
    sta    $FD                       ;
    lda    patpth,y                  ; read pattern pointer high
    sta    $FE                       ; FD-FE: pattern pointer

    lda    #$FF                      ;
    sta    controlSwitch              ; on ADS

    lda    #$00                      ;
    sta    portaFlag,x                ; no portamento
    sta    arpeggioFlag,x             ; no arpeggio
    sta    vibratoFlag,x              ; no vibrato

```

```

; read the pattern value of note to play
readNextPat:
    ldy ptnInd,x          ; load pattern index of this voice
    lda ($FD),y           ; read a pattern value
    cmp #$FD              ; note length
    bcc notFD             ; jump if <$FD

    iny                  ; next pattern value index
    inc ptnInd,x          ; next (saved) pattern value index
    lda ($FD),y           ; read a pattern value (note duration)
    sta noteLength,x      ; store note length duration

incPtnInd:
    inc ptnInd,x          ; next (saved) patter value index
    bne readNextPat

notFD:
    cmp #$FB              ; jump if <$FB
    bcc notFB

    cmp #$FB              ; jump if <>$FB
    bne isFB

isFB:
    lda #$01              ; negative portamento

storePort:
    sta portaFlag,x       ; store in portamento flag
    iny                  ; next pattern value index
    inc ptnInd,x          ; next (saved) pattern value index
    lda ($FD),y           ; read a pattern value
    sta portaVal,x        ; store pattern value (portamento)

    lda #$00
    sta arpeggioFlag,x    ; no arpeggio
    sta vibratoFlag,x     ; no vibrato
    beq incPtnInd

isFC:
    lda #$02              ; positive portamento
    bne storePort        ; store portamento value

notFB:
    cmp #$FA              ; jump if <$FA
    bcc newNote

; select new instruments

    iny                  ; next pattern value index
    inc ptnInd,x          ; next (saved) pattern value index
    lda ($FD),y           ; read a pattern value (instrument index)

    asl
    asl
    asl                  ; index * 8
    sta instrInd,x        ; index of instruments data

    tay
    lda instr1+0,y        ; read Hi/Lo of pulsation amplitude

    pha
    and #$0F
    sta hi1Wave,x         ; Wave form pulsation amplitude (hi byte)
    sta hi2Wave,x         ; Wave form pulsation amplitude (hi byte)
    pla

    and #$F0
    sta lo1Wave,x         ; Wave form pulsation amplitude (lo byte)
    sta lo2Wave,x         ; Wave form pulsation amplitude (lo byte)
    jmp incPtnInd

newNote:
    sta rpNote,x          ; store packet note (readed note to play)

    lda noteLength,x      ; note length duration
    sta actNoteLength,x   ; actual note length duration

    lda #$00
    sta cycleInt,x
    sta cycleEst,x

    lda #$02
    sta dur4Eff,x         ; set duration of effect 4 switch

    ldy instrInd,x        ; index of instrument data
    lda instr1+7,y        ; instrument effect
    and #$02              ; is effect 2?
    beq noEffect2

effect2:
    lda lo2Wave,x         ; Wave form pulsation amplitude (lo byte)

```

```

        sta  lolWave,x                ; Wave form pulsation amplitude (lo byte)

        lda  hi2Wave,x                ; Wave form pulsation amplitude (hi byte)
        sta  hilWave,x                ; Wave form pulsation amplitude (hi byte)

noEffect2:
        lda  rpNote,x                 ; readed note to play
        bne  notZero

        lda  pNote,x                  ; note to play
        sta  rpNote,x                 ; readed note to play
        lda  #$00
        sta  pNote,x                  ; note to play
        ldy  instrInd,x                ; index of instrument data
        dec  controlSwitch
        bne  setVoice

notZero:
        sta  pNote,x                  ; note to play
        tay

        lda  frequencyHi,y
        sta  $D401,x                  ; Voice 1: Frequency control (hi byte)
        sta  freqHilEff,x              ; Voice 1: Frequency control (hi byte) for effect 1
        sta  freqPortHi,x              ; Voice 1: Frequency control (hi byte) for portamento

        lda  frequencyLo,y
        sta  $D400,x                  ; Voice 1: Frequency control (lo byte)
        sta  freqLolEff,x              ; Voice 1: Frequency control (lo byte)
        sta  freqPortLo,x              ; Voice 1: Frequency control (lo byte) for portamento

        ldy  instrInd,x                ; index of instrument data
        lda  instr1+6,y                ; control register (ADS off)
        sta  $D404,x                  ; Voice 1: Control registers

setVoice:
        lda  instr1+1,y                ; control register
        and  controlSwitch             ; change ADS
        sta  $D404,x                  ; Voice 1: Control registers

        lda  instr1+2,y                ; read A/D value
        sta  $D405,x                  ; Generator 1: Attack/Decay

        lda  instr1+3,y                ; read S/R value
        sta  $D406,x                  ; Generator 1: Sustain/Release

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

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

        inc  ptnInd,x                  ; next (saved) pattern value index
        ldy  ptnInd,x                  ; load pattern value index
        lda  ($FD),y                  ; read a pattern value

        cmp  #$FF                      ; end of pattern
        bne  noEndPattern

endPattern:
        lda  #$00
        sta  ptnInd,x                  ; set pattern value index to the beginning

        inc  offsetTr1,x               ; next track position (offset)
        ldy  offsetTr1,x

        lda  ($FB),y                  ; read next track pattern pointer
        cmp  #$FF                      ; end of pattern ?
        bne  noEndTrack

endTrack:
        lda  #$00
        sta  offsetTr1,x               ; set track position to the beginning
        beq  noEndPattern

noEndTrack:
        cmp  #$FE                      ; end of song (no repeat)
        bne  noEndPattern

        lda  #$00
        sta  songsNumber               ; nothing song to play
        rts

noEndPattern:
        lda  pNote,x                  ; note to play
        beq  pulseTimbre
        ldy  instrInd,x                ; index of instrument data
        lda  portaFlag,x               ; portamento flag
        bne  skipSetPort
        lda  instr2+4,y                ; read portamento flag for the instrument

```

```

        beq    skipPortaInstr

        sta    portaFlag,x          ; store portamento flag
        lda    instr2+3,y           ; read portamento value
        sta    portaVal,x           ; portamento value
skipSetPort:
        jmp    testMakePortamento

skipPortaInstr:
        lda    instr1+5,y           ; hi/lo value for arpeggio
        beq    skipArpeggioInit
        jmp    arpeggioInit

skipArpeggioInit:
        sta    arpeggioFlag,x
        lda    instr2+0,y           ; read oscillating frequency value
        beq    noOscEffect
        jmp    vibratoInit

noOscEffect:
        sta    vibratoFlag,x        ; no vibrato
        jmp    decCycleTimer

;=====
; pulse-width timbre routine
;=====
pulseTimbre:
        lda    instr1+4,y           ; Wave amplitude inc/dec value
        sta    ampVar               ; store for late use
        beq    afterWaveShifting

        lda    pulseDirFlag,x
        bne    decWave
        clc
        lda    lolWave,x            ; Wave form pulsation amplitude (lo byte)
        adc    ampVar
        sta    lolWave,x            ; Wave form pulsation amplitude (lo byte)
        sta    $D402,x              ; Voice 1: Wave form pulsation amplitude (lo byte)

        lda    hilWave,x            ; Wave form pulsation amplitude (hi byte)
        adc    #$00
        sta    hilWave,x            ; Wave form pulsation amplitude (hi byte)
        sta    $D403,x              ; Voice 1: Wave form pulsation amplitude (hi byte)

        clc
        cmp    #$0E
        bcc    afterWaveShifting
        inc    pulseDirFlag,x
        bne    afterWaveShifting

decWave:
        lda    lolWave,x            ; Wave form pulsation amplitude (lo byte)
        sec
        sbc    ampVar
        sta    lolWave,x            ; Wave form pulsation amplitude (lo byte)
        sta    $D402,x              ; Voice 1: Wave form pulsation amplitude (lo byte)

        lda    hilWave,x            ; Wave form pulsation amplitude (hi byte)
        sbc    #$00
        sta    hilWave,x            ; Wave form pulsation amplitude (hi byte)
        sta    $D403,x              ; Voice 1: Wave form pulsation amplitude (hi byte)

        clc
        cmp    #$08
        bcs    afterWaveShifting

        dec    pulseDirFlag,x

afterWaveShifting:
        lda    arpeggioFlag,x        ; test for made arpeggio
        beq    testVibratoEff

;=====
; make the arpeggio
;=====
makeArpeggio:
        lda    lowInd,x              ; low index for arpeggio table
        asl
        tay
        lda    pointerLo,y           ; low pointer of arpeggio table
        sta    adcArp+1
        lda    pointerHi,y           ; high pointer of arpeggio table
        sta    adcArp+2

        lda    actualCycle,x
        cmp    limitCycle,x
        bne    noResetCylce

        lda    #$00

```

```

        sta    actualCycle,x

noResetCylce:
        tay
        lda    rpNote,x            ; readed note to play
        clc

adcArp:
        adc    TableArp0,y          ; add the tone to note to play
        tay          ; calciulate the new freq. to use
        lda    frequencyLo,y
        sta    $D400,x              ; Voice 1: Frequency control (lo byte)
        lda    frequencyHi,y
        sta    $D401,x              ; Voice 1: Frequency control (hi byte)
        inc    actualCycle,x
        jmp    decCycleTimer

testVibratoEff:
        lda    vibratoFlag,x        ; load vibrato flag
        bne    vibratoEffect        ; is to performe vibrato?
        jmp    testMakePortamento

;=====
; make the vibrato
;=====
vibratoEffect:
        lda    countVibFlag,x
        beq    sbcVib
        cmp    #$03
        bcc    adcVib

        sec
        lda    freqPortLo,x          ; Voice 1: Frequency control (lo byte) for portamento
        sbc    oscilFreq,x           ; frequency of oscillation
        sta    freqPortLo,x          ; Voice 1: Frequency control (lo byte) for portamento
        sta    $D400,x               ; Voice 1: Frequency control (lo byte)

        lda    freqPortHi,x          ; Voice 1: Frequency control (hi byte) for portamento
        sbc    #$00
        sta    freqPortHi,x          ; Voice 1: Frequency control (hi byte) for portamento
        sta    $D401,x               ; Voice 1: Frequency control (hi byte)

        dec    actualVibCount,x
        bne    exitVib1

        lda    vibCount,x            ; read stored value of count
        sta    actualVibCount,x       ; set to actual
        inc    countVibFlag,x
        lda    countVibFlag,x
        cmp    #$05
        bcc    exitVib1

        lda    #$01
        sta    countVibFlag,x
exitVib1:
        jmp    decCycleTimer

sbcVib:
        sec
        lda    freqPortLo,x          ; Voice 1: Frequency control (lo byte) for portamento
        sbc    oscilFreq,x           ; frequency of oscillation
        sta    freqPortLo,x          ; Voice 1: Frequency control (lo byte) for portamento
        sta    $D400,x               ; Voice 1: Frequency control (lo byte)

        lda    freqPortHi,x          ; Voice 1: Frequency control (hi byte) for portamento
        sbc    #$00
        sta    freqPortHi,x          ; Voice 1: Frequency control (hi byte) for portamento
        sta    $D401,x               ; Voice 1: Frequency control (hi byte)

        dec    actualVibCount,x
        bne    exitVib2

        lda    vibCount,x            ; read stored count value
        sta    actualVibCount,x
        inc    countVibFlag,x
exitVib2:
        jmp    decCycleTimer

adcVib:
        clc
        lda    freqPortLo,x          ; Voice 1: Frequency control (lo byte) for portamento
        adc    oscilFreq,x           ; frequency of oscillation
        sta    freqPortLo,x          ; Voice 1: Frequency control (lo byte) for portamento
        sta    $D400,x               ; Voice 1: Frequency control (lo byte)

        lda    freqPortHi,x          ; Voice 1: Frequency control (hi byte) for portamento
        adc    #$00
        sta    freqPortHi,x          ; Voice 1: Frequency control (hi byte) for portamento
        sta    $D401,x               ; Voice 1: Frequency control (hi byte)

        dec    actualVibCount,x

```



```

    bne    noEffect1

    lda    vibCount,x          ; read stored count value
    sta    actualVibCount,x
    inc    countVibFlag,x
    bne    noEffect1
    jmp    decCycleTimer

testMakePortamento:
    lda    portaFlag,x         ; portamento flag
    beq    testEffect1

    cmp    #$01                ; negative portamento
    beq    negativePortamento

    cmp    #$02                ; positive portamento
    beq    positivePortamento

    cmp    #$03
    beq    negativePortamentoHI
    clc
    lda    freqPortHi,x        ; Voice 1: Frequency control (hi byte) for portamento
    adc    portaVal,x          ; add portamento value
    sta    freqPortHi,x        ; Voice 1: Frequency control (hi byte) for portamento
    sta    $D401,x             ; Voice 1: Frequency control (hi byte)
    jmp    testEffect1

negativePortamento:
    clc
    lda    freqPortLo,x        ; Voice 1: Frequency control (lo byte) for portamento
    sbc    portaVal,x          ; sub portamento value
    sta    freqPortLo,x        ; Voice 1: Frequency control (lo byte) for portamento
    sta    $D400,x             ; Voice 1: Frequency control (lo byte)

    lda    freqPortHi,x        ; Voice 1: Frequency control (hi byte) for portamento
    sbc    #$00
    sta    freqPortHi,x        ; Voice 1: Frequency control (hi byte) for portamento
    sta    $D401,x             ; Voice 1: Frequency control (hi byte)
    jmp    testEffect1

negativePortamentoHI:
    sec
    lda    freqPortHi,x        ; Voice 1: Frequency control (hi byte) for portamento
    sbc    portaVal,x          ; sub portamento value
    sta    freqPortHi,x        ; Voice 1: Frequency control (hi byte) for portamento
    sta    $D401,x             ; Voice 1: Frequency control (hi byte)
    jmp    testEffect1

positivePortamento:
    clc
    lda    freqPortLo,x        ; Voice 1: Frequency control (lo byte) for portamento
    adc    portaVal,x          ; add portamento value
    sta    freqPortLo,x        ; Voice 1: Frequency control (lo byte) for portamento
    sta    $D400,x             ; Voice 1: Frequency control (lo byte)

    lda    freqPortHi,x        ; Voice 1: Frequency control (hi byte) for portamento
    adc    #$00
    sta    freqPortHi,x        ; Voice 1: Frequency control (hi byte) for portamento
    sta    $D401,x             ; Voice 1: Frequency control (hi byte)

testEffect1:
    ldy    instrInd,x          ; index of instrument data
    lda    instr1+7,y          ; load instrument effect
    and    #$01                ; is effect 1?
    beq    noEffect1
    jmp    effect1

noEffect1:
    jmp    decCycleTimer

;=====
; Voice 1
;
; vibrato flag: 1= on
;
; arpeggio flag: 1= on
;
; portaFlag:
; 1: neg. port. HiLo
; 2: pos. port. HiLo
; 3: neg. port. Hi
; x: pos. port. Hi
;
; pulseDirFlag: 1 dec, 0 inc
;=====

vibratoFlag:                ; vibrato flag (1=on)
    .byte    $00

arpeggioFlag:                ; 1= effect on

```

```

.byte $00

portaFlag:
.byte $00 ; portamento flag (voice 1)

cycleInt:
.byte $00 ; internal cycle for effect 1

cycleEst:
.byte $00 ; external cycle for effect 1

ptnInd:
ptnInd1:
.byte $06 ; pattern value index (voice 1)

pulseDirFlag:
.byte $00

;=====
; Voice 2:
;=====
.byte $00
.byte $00
.byte $00 ; portamento flag (voice 2)
.byte $00
.byte $00
ptnInd2:
.byte $06 ; pattern value index (voice 2)
.byte $00

;=====
; Voice 3:
;=====
.byte $00
.byte $00
.byte $00 ; portamento flag
.byte $00
.byte $00
ptnInd3:
.byte $00 ; pattern value index (voice 3)
.byte $01

;=====
; Voice 1:
;=====
portaVal:
.byte $00 ; portamento value (voice 1)

r0CE4:
.byte $00 ; not used

noteLength:
.byte $3F ; note length duration (voice 1)

instrInd:
.byte $08 ; index of instrument data (voice 1)

lo1Wave:
.byte $BB ; Wave form pulsation amplitude (lo byte)

lo2Wave:
.byte $90 ; Wave form pulsation amplitude (lo byte)

hi1Wave:
.byte $02 ; Wave form pulsation amplitude (hi byte)

;=====
; Voice 2:
;=====
.byte $00 ; portamento value (Voice 2)
.byte $00
.byte $3F ; note length duration (voice 2)
.byte $08 ; index of instrument data (voice 2)
.byte $BB ; Wave form pulsation amplitude (lo byte)
.byte $90 ; Wave form pulsation amplitude (lo byte)
.byte $02 ; Wave form pulsation amplitude (hi byte)

;=====
; Voice 3:
;=====
.byte $00 ; portamento value (voice 3)
.byte $00
.byte $3F ; note length duration (voice 3)
.byte $20 ; index of instrument data (voice 3)
.byte $F0 ; Wave form pulsation amplitude (lo byte)
.byte $90 ; Wave form pulsation amplitude (lo byte)
.byte $0C ; Wave form pulsation amplitude (hi byte)

```

```

;=====
; Voice 1:
;=====
hi2Wave:
    .byte $00                ; Wave form pulsation amplitude (hi byte)

lowInd:
    .byte $00                ; low index for freq. instrument effect

currentTr1:
    .byte <drillerMainTr1    ; current track 1 position (base)
    .byte >drillerMainTr1

offsetTr1:
    .byte $00                ; current track 1 position (offset)

r0CFD:
    .byte $00                ; not used

actNoteLength =*
    .byte $3C                ; actual note length duration

;=====
; Voice2:
;=====
    .byte $00
    .byte $00                ; low index for freq. instrument effect
currentTr2:
    .byte <drillerMainTr2    ; current track 2 position (base)
    .byte >drillerMainTr2
offsetTr2:
    .byte $00                ; current track 2 position (offset)
    .byte $00
    .byte $3C                ; actual note length duration

;=====
; Voice 3:
;=====
    .byte $06
    .byte $00                ; low index for freq. instrument effect
currentTr3:
    .byte <drillerMainTr3    ; current track 3 position (base)
    .byte >drillerMainTr3
offsetTr3:
    .byte $02                ; current track 3 position (offset)
    .byte $00
    .byte $3C                ; actual note length duration

r0D0D:
    .byte $00
    .byte $00

songsNumber =*
    .byte $AB                ; 0=nothing 1=first, $AB current ?

speed:
    .byte $03                ; cycle timer (cylce for having 1 note tick)

ampVar:
    .byte $A0                ; wave amplitude variation

actSpeed:
    .byte $00                ; actual cycle timer (for having 1 note tick)

controlSwitch:
    .byte $FE                ; set ON/OFF the ADS

;=====
; Voice 1
;=====

freqPortLo:
    .byte $47                ; frequency of portamento lo
                                ; Voice 1: Frequency control (lo byte)

freqLo1Eff:
    .byte $47                ; freq lo for instrument effect 1 ?
                                ; Voice 1: Frequency control (lo byte)

freqHi1Eff:
    .byte $06                ; freq hi for instrument effect 1
                                ; Voice 1: Frequency control (hi byte)

rpNote =*
    .byte $1F                ; Voice 1: readed note to play

freqPortHi:
    .byte $06                ; frequency of portamento Hi
                                ; Voice 1: Frequency control (hi byte)

limitCycle:
    .byte $00                ; limit of cycle for arpeggio

```

```

actualCycle:                                ; actual cycle for arpeggio
.byte $00

;=====
; Voice 2
;=====
.byte $23                                ; frequency of portamento lo
.byte $23                                ; freq lo for instrument effect 1 ?
.byte $03                                ; freq hi for instrument effect 1
.byte $13                                ; Voice 2: readed note to play
.byte $03                                ; frequency of portamento Hi
.byte $00                                ; limit of cycle for arpeggio
.byte $00                                ; actual cycle for arpeggio

;=====
; Voice 3
;=====

.byte $00                                ; frequency of portamento lo
.byte $00                                ; freq lo for instrument effect 1 ?
.byte $00                                ; freq hi for instrument effect 1
.byte $00                                ; Voice 3: readed note to play
.byte $00                                ; frequency of portamento Hi
.byte $00

;=====
; Voice 1
;=====

countVibFlag:                              ; counter flag: use for add/sbc of vibrato
.byte $00

oscilFreq:                                ; frequency of oscillator (for vibrato)
.byte $00

vibCount:                                  ; value of vibrato count
.byte $00

actualVibCount:                            ; actual value of vibrato count intensity
.byte $00

r0D2D:
.byte $00
.byte $00

pNote =*
.byte $1F                                ; note to play

;=====
; Voice 2
;=====
.byte $00
.byte $00
.byte $00
.byte $00
.byte $00
.byte $00
.byte $13                                ; note to play

;=====
; Voice 3
;=====
.byte $00
.byte $00
.byte $00
.byte $00
.byte $00
.byte $00
.byte $00                                ; note to play

;=====
; Voice 1
;=====

dur4Eff:
.byte $02                                ; duration of effect 4 switch
.byte $00
.byte $00
.byte $00
.byte $00
.byte $00
.byte $00

;=====
; Voice 2
;=====
.byte $02
.byte $00

```

```

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

;=====
; Voice 3
;=====
        .byte $02
        .byte $00
        .byte $00
        .byte $00
        .byte $00
        .byte $00
        .byte $00

; freq. low

frequencyLo:
        .byte $0C, $1C, $2D, $3E      ; C-0 C#-0 D-0 D#-0
        .byte $51, $66, $7B, $91
        .byte $A9, $C3, $DD, $FA
        .byte $18, $38, $5A, $7D
        .byte $A3, $CC, $F6, $23
        .byte $53, $86, $BB, $F4
        .byte $30, $70, $B4, $FB
        .byte $47, $98, $ED, $47
        .byte $A7, $0C, $77, $E9
        .byte $61, $E1, $68, $F7
        .byte $8F, $30, $DA, $8F
        .byte $4E, $18, $EF, $D2
        .byte $C3, $C3, $D1, $EF
        .byte $1F, $60, $B5, $1E
        .byte $9C, $31, $DF, $A5
        .byte $87, $86, $A2, $DF
        .byte $3E, $C1, $6B, $3C
        .byte $39, $63, $BE, $4B
        .byte $0F, $0C, $45, $BF
        .byte $7D, $83, $D6, $79
        .byte $73, $C7, $7C, $97
        .byte $1E, $18, $8B, $7E
        .byte $FA, $06, $AC, $F3
        .byte $E6, $8F, $F8, $2E

frequencyHi:
        .byte $01, $01, $01, $01
        .byte $01, $01, $01, $01
        .byte $01, $01, $01, $01
        .byte $02, $02, $02, $02
        .byte $02, $02, $02, $03
        .byte $03, $03, $03, $03
        .byte $04, $04, $04, $04
        .byte $05, $05, $05, $06
        .byte $06, $07, $07, $07
        .byte $08, $08, $09, $09
        .byte $0A, $0B, $0B, $0C
        .byte $0D, $0E, $0E, $0F
        .byte $10, $11, $12, $13
        .byte $15, $16, $17, $19
        .byte $1A, $1C, $1D, $1F
        .byte $21, $23, $25, $27
        .byte $2A, $2C, $2F, $32
        .byte $35, $38, $3B, $3F
        .byte $43, $47, $4B, $4F
        .byte $54, $59, $5E, $64
        .byte $6A, $70, $77, $7E
        .byte $86, $8E, $96, $9F
        .byte $A8, $B3, $BD, $C8
        .byte $D4, $E1, $EE, $FD

activeCiaIrq =*
        sei
        lda #<serviceIrq
        sta $0314      ; Vector: Hardware Interrupt (IRQ)
        lda #>serviceIrq
        sta $0315      ; Vector: Hardware Interrupt (IRQ)
        ldx #$00
        stx $DC0E      ; Control register A of CIA #1
        inx
        stx $D01A      ; IRQ mask register
        cli
        rts

serviceIrq =*
        lda #$01
        sta $D019      ; Interrupt indicator register
        lda #$82
        sta $D012      ; Reading/Writing IRQ balance value
        lda #$1B

```

Definition of the used frequencies

```

        sta $D011                ; VIC control register
        lda #$01
        sta $D020                ; Border color
        jsr serviceMusic
        dec $D020                ; Border color
        jmp $EA31                ; Default hardware interrupt (IRQ)

serviceMusic =*
        ldx #$00                ; Voice 1
        jsr makeMusic
        ldx #$07                ; Voice 2
        jsr makeMusic
        ldx #$0E                ; Voice 3
        jsr makeMusic
        rts

        .byte "(C)1987 MATT GRAY"

;=====
; Init the arpeggio routine
;=====
arpeggioInit:
        pha
        and #$0F
        sta lowInd,x
        pla
        and #$F0
        lsr a
        lsr a
        lsr a
        lsr a
        sta limitCycle,x

        lda #$00
        sta actualCycle,x

        lda #$01                ; made on arpeggio
        sta arpeggioFlag,x

        lda #$00
        sta vibratoFlag,x        ; no vibrato
        jmp decCycleTimer

;=====
; Init the vibrato routine
;=====
vibratoInit:
        sta oscilFreq,x          ; oscillating frequency value
        lda instr2+1,y           ; read the count intensity for vibrato
        sta vibCount,x           ; save count value
        sta actualVibCount,x
        lda #$00
        sta arpeggioFlag,x        ; make arpeggio off
        sta countVibFlag,x

        lda #$01
        sta vibratoFlag,x        ; make vibrato
        jmp decCycleTimer

;=====
; instruments part 1
; 0: Hi/Lo of Wave form amplitude
; 1: Control register
; 2: A/D value
; 3: S/R value
; 4: Wave amplitude inc/dec value
; 5: hi/lo value for arpeggio
; 6: Control register
; 7: instrument effect
; 1: a frequency effect
; 2: a pulse wave effect
; 4: switch between waveform
;=====

instr1 =*
        .byte $00                ; Wave form amplitude (Hi/Lo)
        .byte $81                ; Control: Noise, ADS on
        .byte $0A                ; A/D
        .byte $00                ; S/R
        .byte $00
        .byte $00
        .byte $80                ; Control: Noise, ADS off
        .byte $01                ; instrument effect

        .byte $90
        .byte $41
        .byte $FE
        .byte $0D
        .byte $25
        .byte $00

```

The first table of
instruments definition

```

.byte $40
.byte $02

.byte $00
.byte $81
.byte $FD
.byte $00
.byte $00
.byte $00
.byte $80
.byte $00

.byte $30
.byte $41
.byte $0E
.byte $00
.byte $30
.byte $00
.byte $40
.byte $02

.byte $96
.byte $41
.byte $0E
.byte $00
.byte $A0
.byte $00
.byte $40
.byte $02

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

.byte $32
.byte $41
.byte $00
.byte $40
.byte $F0
.byte $00
.byte $40
.byte $02

.byte $00
.byte $81
.byte $08
.byte $00
.byte $00
.byte $00
.byte $80
.byte $01

.byte $00
.byte $11
.byte $0D
.byte $00
.byte $00
.byte $00
.byte $10
.byte $00

.byte $90
.byte $41
.byte $0E
.byte $00
.byte $25
.byte $00
.byte $40
.byte $02

.byte $2E
.byte $43
.byte $00
.byte $60
.byte $F5
.byte $00
.byte $40
.byte $04

.byte $70
.byte $41
.byte $0A
.byte $00
.byte $40
.byte $00

```

```

.byte $40
.byte $02

.byte $00
.byte $15
.byte $03
.byte $00
.byte $00
.byte $20
.byte $14
.byte $04

.byte $40
.byte $41
.byte $00
.byte $90
.byte $01
.byte $00
.byte $40
.byte $00

.byte $00
.byte $15
.byte $EE
.byte $00
.byte $00
.byte $00
.byte $14
.byte $00

.byte $98
.byte $41
.byte $09
.byte $00
.byte $00
.byte $00
.byte $40
.byte $01

.byte $21
.byte $41
.byte $0A
.byte $00
.byte $30
.byte $00
.byte $40
.byte $06

.byte $21
.byte $41
.byte $0A
.byte $00
.byte $30
.byte $00
.byte $40
.byte $06

.byte $31
.byte $41
.byte $0E
.byte $00
.byte $10
.byte $00
.byte $40
.byte $02

.byte $23
.byte $41
.byte $00
.byte $A0
.byte $50
.byte $00
.byte $40
.byte $00

.byte $91
.byte $41
.byte $0A
.byte $00
.byte $30
.byte $00
.byte $40
.byte $06

.byte $F1
.byte $41
.byte $0C
.byte $00
.byte $40
.byte $00

```



```

    .byte $40
    .byte $06

;=====
; instruments part 2
; 0: oscillating frequency value (for vibrato)
; 1: length of vibrato intensity (for vibrato)
; 2: Control register for effect 4
; 3: portamento value
; 4: portamento flag (1=on) | portamento for the instrument
; 5: duration cycle for effect 1
; 6: not used
; 7: not used
;=====

instr2 =*
    .byte $00
    .byte $00
    .byte $11                ; Control: ^, on ADS
    .byte $00                ; portamento value
    .byte $00                ; portamento flag
    .byte $03
    .byte $00                ; not used
    .byte $00                ; not used

    .byte $00
    .byte $00
    .byte $81
    .byte $00
    .byte $00
    .byte $00
    .byte $00
    .byte $00

    .byte $06
    .byte $50
    .byte $00
    .byte $00
    .byte $00
    .byte $00
    .byte $00
    .byte $00

    .byte $30
    .byte $02
    .byte $81
    .byte $00
    .byte $00
    .byte $00
    .byte $00
    .byte $00

    .byte $40
    .byte $02
    .byte $00
    .byte $00
    .byte $00
    .byte $00
    .byte $00
    .byte $00

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

    .byte $00
    .byte $00
    .byte $81
    .byte $00
    .byte $00
    .byte $00
    .byte $00
    .byte $00

    .byte $00
    .byte $00
    .byte $11
    .byte $41
    .byte $01
    .byte $01
    .byte $00
    .byte $00

    .byte $50
    .byte $02

```

The second table for
instruments definition

.byte \$00
.byte \$00
.byte \$00
.byte \$00
.byte \$00
.byte \$00

.byte \$00
.byte \$00
.byte \$81
.byte \$00
.byte \$00
.byte \$00
.byte \$00
.byte \$00

.byte \$20
.byte \$02
.byte \$00
.byte \$00
.byte \$00
.byte \$00
.byte \$00
.byte \$00

.byte \$00
.byte \$00
.byte \$00
.byte \$00
.byte \$00
.byte \$00
.byte \$00
.byte \$00

.byte \$00
.byte \$00
.byte \$81
.byte \$00
.byte \$00
.byte \$00
.byte \$00
.byte \$00
.byte \$00

.byte \$40
.byte \$02
.byte \$00
.byte \$00
.byte \$00
.byte \$00
.byte \$00
.byte \$00
.byte \$00

.byte \$00
.byte \$00
.byte \$00
.byte \$00
.byte \$00
.byte \$00
.byte \$00
.byte \$00

.byte \$00
.byte \$00
.byte \$41
.byte \$F0
.byte \$01
.byte \$01
.byte \$00
.byte \$00

.byte \$10
.byte \$02
.byte \$43
.byte \$00
.byte \$00
.byte \$00
.byte \$00
.byte \$00

.byte \$00
.byte \$00
.byte \$00
.byte \$00
.byte \$00
.byte \$00
.byte \$00
.byte \$00

.byte \$A0
.byte \$02

```

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

.byte $60
.byte $02
.byte $00
.byte $00
.byte $00
.byte $00
.byte $00
.byte $00

.byte $00
.byte $00
.byte $43
.byte $00
.byte $00
.byte $00
.byte $00
.byte $00

.byte $0A
.byte $02
.byte $43
.byte $00
.byte $00
.byte $00
.byte $00

effect1:
    lda    freqHilEff,x        ; Voice 1: Frequency control (hi byte) for effect 1
    beq    noDec
    dec    freqHilEff,x        ; Voice 1: Frequency control (hi byte) for effect 1
noDec:
    lda    cycleInt,x
    beq    jmpTestEndCycle

    dec    cycleInt,x
    lda    #$81                ; noise + ADS
    sta    $D404,x             ; Voice 1: Control registers
    lda    freqHilEff,x        ; Voice 1: Frequency control (hi byte) for effect 1
    eor    #$23
    sta    $D401,x             ; Voice 1: Frequency control (hi byte)
    jmp    decCycleTimer

jmpTestEndCycle:
    jmp    testEndCycle

changeFreq:
    lda    freqPortHi,x        ; Voice 1: Frequency control (hi byte) for portamento
    sta    $D401,x             ; Voice 1: Frequency control (hi byte)
    sta    freqHilEff,x        ; Voice 1: Frequency control (hi byte) for effect 1
    lda    instr2+2,y          ; control register
    sta    $D404,x             ; Voice 1: Control registers
    jmp    decCycleTimer

testEndCycle:
    lda    cycleEst,x
    cmp    instr2+5,y          ; duration cycle for effect 1
    beq    resetCycle

    inc    cycleInt,x
    inc    cycleEst,x
    bne    changeFreq

resetCycle:
    lda    #$00
    sta    cycleEst,x
    sta    cycleInt,x
    beq    changeFreq

songSpeed =*                  ; cycle for having note tick (for each songs)
.byte $00
.byte $03
.byte $03

; $FF: repeat the track
; $FE: end of music

drillerMainTrl:
    .byte $01, $01, $07, $09
    .byte $09, $09, $01, $07
    .byte $07, $0F, $0F, $0F
    .byte $0F, $0F, $0F, $03
    .byte $03, $0F, $0F, $13

```

The definition of the patterns used by each traks are from here

```

.byte $13, $0F, $13, $0F
.byte $13, $0F, $13, $0F
.byte $13, $0F, $0F, $0F
.byte $0F, $0F, $0F, $0F
.byte $0F, $0F, $0F, $0F
.byte $0F, $0F, $0F, $0F
.byte $0F, $0F, $0F, $0F
.byte $0F, $0F, $0F, $0F
.byte $0F, $0F, $1B, $1D
.byte $1E, $0F, $1B, $1D
.byte $1E, $0F, $1B, $1D
.byte $1E, $12, $12, $12
.byte $12, $24, $24, $21
.byte $21, $24, $24, $21
.byte $21, $24, $24, $21
.byte $21, $24, $24, $21
.byte $21, $24, $24, $21
.byte $21, $24, $24, $21
.byte $21, $24, $24, $21
.byte $21, $24, $24, $21
.byte $21, $24, $24, $21
.byte $21, $24, $24, $21
.byte $21, $24, $24, $21
.byte $21, $08, $08, $28
.byte $00, $00, $00, $00
.byte $00, $FF

drillerMainTr2:
.byte $03, $03, $08, $0A
.byte $0D, $0D, $0D, $0D
.byte $08, $07, $0E, $0E
.byte $0E, $0E, $0E, $0E
.byte $0E, $0E, $05, $12
.byte $12, $12, $12, $14
.byte $15, $14, $15, $14
.byte $15, $14, $15, $08
.byte $17, $17, $17, $17
.byte $17, $17, $17, $17
.byte $17, $17, $17, $17
.byte $07, $07, $1F, $1F
.byte $1F, $1F, $07, $07
.byte $00, $00, $25, $25
.byte $26, $25, $27, $27
.byte $27, $27, $27, $27
.byte $27, $27, $06, $06
.byte $06, $06, $06, $06
.byte $06, $06, $06, $06
.byte $28, $00, $00, $00
.byte $00, $FF

drillerMainTr3:
.byte $00, $00, $00, $00
.byte $04, $06, $06, $0C
.byte $0B, $0C, $0B, $0C
.byte $0B, $06, $06, $06
.byte $06, $06, $06, $06
.byte $06, $06, $06, $06
.byte $06, $06, $06, $0F
.byte $0F, $10, $11, $0E
.byte $0E, $0E, $0E, $0E
.byte $0E, $0E, $0E, $0E
.byte $0E, $0E, $0E, $16
.byte $07, $07, $07, $18
.byte $19, $19, $1A, $1A
.byte $08, $08, $1C, $08
.byte $08, $23, $23, $22
.byte $22, $23, $23, $22
.byte $22, $23, $23, $22
.byte $22, $23, $23, $22
.byte $22, $23, $23, $22
.byte $22, $23, $23, $22
.byte $22, $23, $23, $22
.byte $22, $23, $23, $22
.byte $22, $23, $23, $22
.byte $22, $23, $23, $22
.byte $22, $23, $23, $22
.byte $22, $23, $23, $22
.byte $22, $23, $23, $22
.byte $22, $07, $07, $0F
.byte $0F, $0F, $0F, $29
.byte $00, $00, $00, $00
.byte $FF

;=====
; pattern data
;=====

; format:
; 00          : release gate
; xx          : note xx
; $FA nn      : select instrument nn
; $FB mm      : negative portamento (mm)
; $FC kk      : positive portamento (kk)
; $FD kk      : duration kk

```

```
; $FF : end of pattern
```

```
pnt00 =*  
    .byte $FD, $3F  
    .byte $FA, $04  
    .byte $00  
    .byte $FF  
  
pnt01 =*  
    .byte $FA, $01  
    .byte $FD, $3F  
    .byte $23, $1F, $22, $1E  
    .byte $FF  
  
pnt03 =*  
    .byte $FA, $01  
    .byte $FD, $3F  
    .byte $17, $13, $16, $12  
    .byte $FF  
  
pnt02 =*  
    .byte $FD, $0F  
    .byte $FA, $04  
    .byte $00  
    .byte $FF  
  
pnt04 =*  
    .byte $FA, $02  
    .byte $FD, $7F  
    .byte $25, $25  
    .byte $FF  
  
pnt05 =*  
    .byte $FA, $0E  
    .byte $FD, $3F  
    .byte $2F, $2B, $2E  
    .byte $FC, $20  
    .byte $2A  
    .byte $FF  
  
pnt06 =*  
    .byte $FA, $06  
    .byte $FD, $01  
    .byte $42, $3B, $3B, $42  
    .byte $3B, $3B, $43, $3B  
    .byte $42, $3B, $3B, $42  
    .byte $3B, $3B, $43, $3B  
    .byte $42, $3B, $3B, $42  
    .byte $3B, $3B, $43, $3B  
    .byte $42, $3B, $3B, $42  
    .byte $3B, $3B, $43, $3B  
    .byte $FF  
  
pnt07 =*  
    .byte $FA, $01  
    .byte $FD, $7F  
    .byte $23  
    .byte $FF  
  
pnt08 =*  
    .byte $FA, $01  
    .byte $FD, $7F  
    .byte $17, $00  
    .byte $FF  
  
pnt09 =*  
    .byte $FA, $09  
    .byte $FD, $1F  
    .byte $17, $13, $12, $0F  
    .byte $FF  
  
pnt0A =*  
    .byte $FA, $08  
    .byte $FD, $0F  
    .byte $3E, $39  
    .byte $FD, $1F  
    .byte $3B  
    .byte $FD, $0F  
    .byte $3D, $3B  
    .byte $FD, $1F  
    .byte $3A  
    .byte $FD, $7F  
    .byte $FB, $01  
    .byte $2F  
    .byte $FF  
  
pnt0B =*  
    .byte $FA, $06  
    .byte $FD, $01
```

All the patterns of
notes start from here

```

        .byte $3D, $36, $36, $3D
        .byte $36, $36, $3E, $36
        .byte $3D, $36, $36, $3D
        .byte $36, $36, $3E, $36
        .byte $3A, $33, $33, $3A
        .byte $33, $33, $3B, $33
        .byte $3A, $33, $33, $3A
        .byte $33, $33, $3B, $33
        .byte $FF

pnt0C =*
        .byte $FA, $06
        .byte $FD, $01
        .byte $42, $3B, $3B, $42
        .byte $3B, $3B, $43, $3B
        .byte $42, $3B, $3B, $42
        .byte $3B, $3B, $43, $3B
        .byte $3E, $37, $37, $3E
        .byte $37, $37, $3F, $37
        .byte $3E, $37, $37, $3E
        .byte $37, $37, $3F, $37
        .byte $FF

pnt0D =*
        .byte $FA, $0A
        .byte $FD, $01
        .byte $3B, $3A, $39, $38
        .byte $39, $3A, $3B, $3A
        .byte $39, $38, $39, $3A
        .byte $3B, $3A, $39, $38
        .byte $39, $3A, $3B, $3A
        .byte $39, $38, $39, $3A
        .byte $3B, $3A, $39, $38
        .byte $39, $3A, $3B, $3A
        .byte $FF

pnt0E =*
        .byte $FA, $07
        .byte $FD, $01
        .byte $2D
        .byte $FD, $03
        .byte $2D
        .byte $FD, $0D
        .byte $2D
        .byte $FD, $03
        .byte $2D
        .byte $FD, $07
        .byte $FA, $00
        .byte $2D
        .byte $FA, $07
        .byte $FD, $01
        .byte $2D
        .byte $FD, $03
        .byte $2D
        .byte $FD, $0D
        .byte $2D
        .byte $FD, $03
        .byte $2D
        .byte $FD, $07
        .byte $FA, $00
        .byte $2D
        .byte $FF

pnt0F =*
        .byte $FA, $0B
        .byte $FD, $01
        .byte $23, $23, $23, $23
        .byte $23, $23, $23, $23
        .byte $23, $23, $23, $23
        .byte $23, $23, $23, $23
        .byte $23, $23, $23, $23
        .byte $23, $23, $23, $23
        .byte $23, $23, $23, $23
        .byte $23, $23, $23, $23
        .byte $23, $23, $23, $23
        .byte $FF

pnt10 =*
        .byte $FA, $0B
        .byte $FD, $01
        .byte $22, $22, $22, $22
        .byte $22, $22, $22, $22
        .byte $22, $22, $22, $22
        .byte $22, $22, $22, $22
        .byte $22, $22, $22, $22
        .byte $22, $22, $22, $22
        .byte $22, $22, $22, $22
        .byte $22, $22, $22, $22
        .byte $22, $22, $22, $22
        .byte $FF

pnt11 =*

```

```

        .byte $FA, $0B
        .byte $FD, $01
        .byte $25, $25, $25, $25
        .byte $25, $25, $25, $25
        .byte $25, $25, $25, $25
        .byte $25, $25, $25, $25
        .byte $25, $25, $25, $25
        .byte $25, $25, $25, $25
        .byte $25, $25, $25, $25
        .byte $25, $25, $25, $25
        .byte $FF

pnt12 =*
        .byte $FA, $0A
        .byte $FD, $01
        .byte $3B, $37, $36, $34
        .byte $3B, $37, $36, $34
        .byte $3B, $37, $36, $34
        .byte $3B, $37, $36, $34
        .byte $3B, $37, $36, $34
        .byte $3B, $37, $36, $34
        .byte $3B, $37, $36, $34
        .byte $3B, $37, $36, $34
        .byte $3B, $37, $36, $34
        .byte $FF

pnt13 =*
        .byte $FA, $0B
        .byte $FD, $01
        .byte $1F, $1F, $1F, $1F
        .byte $1F, $1F, $1F, $1F
        .byte $1F, $1F, $1F, $1F
        .byte $1F, $1F, $1F, $1F
        .byte $1F, $1F, $1F, $1F
        .byte $1F, $1F, $1F, $1F
        .byte $1F, $1F, $1F, $1F
        .byte $1F, $1F, $1F, $1F
        .byte $1F, $1F, $1F, $1F
        .byte $FF

pnt14 =*
        .byte $FA, $06
        .byte $FD, $01
        .byte $3F, $3B, $36, $3F
        .byte $3B, $36, $3F, $3B
        .byte $3F, $3B, $36, $3F
        .byte $3B, $36, $3F, $3B
        .byte $3F, $3B, $36, $3F
        .byte $3B, $36, $3F, $3B
        .byte $3F, $3B, $36, $3F
        .byte $3B, $36, $3F, $3B
        .byte $FF

pnt15 =*
        .byte $FA, $06
        .byte $FD, $01
        .byte $3E, $3B, $37, $3E
        .byte $3B, $37, $3E, $3B
        .byte $3E, $3B, $37, $3E
        .byte $3B, $37, $3E, $3B
        .byte $3E, $3B, $37, $3E
        .byte $3B, $37, $3E, $3B
        .byte $3E, $3B, $37, $3E
        .byte $3B, $37, $3E, $3B
        .byte $FF

pnt16 =*
        .byte $FA, $0D
        .byte $FD, $1F
        .byte $37, $36, $39, $37
        .byte $36, $2F, $2F, $32
        .byte $FF

pnt17 =*
        .byte $FA, $10
        .byte $FD, $01
        .byte $23, $23, $2A, $2A
        .byte $28, $28, $2A, $2A
        .byte $26, $26, $2A, $2A
        .byte $28, $28, $2A, $2A
        .byte $23, $23, $2A, $2A
        .byte $28, $28, $2A, $2A
        .byte $26, $26, $2A, $2A
        .byte $28, $28, $2A, $2A
        .byte $FF

pnt18 =*
        .byte $FA, $13
        .byte $FD, $07
        .byte $FC, $37, $45
        .byte $FD, $2F
        .byte $47

```

```

    .byte $FD, $07
    .byte $FB, $7F, $47
    .byte $FD, $37
    .byte $42
    .byte $FD, $07
    .byte $FB, $80
    .byte $42
    .byte $FF

pnt19 =*
    .byte $FA, $13
    .byte $FD, $1F
    .byte $3B
    .byte $FD, $0F
    .byte $39, $37
    .byte $FD, $3F
    .byte $36
    .byte $FF

pnt1A =*
    .byte $FA, $13
    .byte $FD, $1F
    .byte $34
    .byte $FD, $0F
    .byte $32, $31
    .byte $FD, $3F
    .byte $2F
    .byte $FF

pnt1B =*
    .byte $FA, $0B
    .byte $FD, $01
    .byte $1B, $1B, $1B, $1B
    .byte $1B, $1B, $1B, $1B
    .byte $1B, $1B, $1B, $1B
    .byte $1B, $1B, $1B, $1B
    .byte $1B, $1B, $1B, $1B
    .byte $1B, $1B, $1B, $1B
    .byte $1B, $1B, $1B, $1B
    .byte $1B, $1B, $1B, $1B
    .byte $FF

pnt1C =*
    .byte $FA, $01
    .byte $FD, $1F
    .byte $3B
    .byte $FD, $0F
    .byte $3A, $36
    .byte $FD, $2F
    .byte $36
    .byte $FD, $0F
    .byte $38
    .byte $FD, $1F
    .byte $38, $2F, $31
    .byte $FD, $0F
    .byte $33, $34
    .byte $FD, $7F
    .byte $36, $36
    .byte $FF

pnt1D =*
    .byte $FA, $0B
    .byte $FD, $01
    .byte $1C, $1C, $1C, $1C
    .byte $1C, $1C, $1C, $1C
    .byte $1C, $1C, $1C, $1C
    .byte $1C, $1C, $1C, $1C
    .byte $1C, $1C, $1C, $1C
    .byte $1C, $1C, $1C, $1C
    .byte $1C, $1C, $1C, $1C
    .byte $1C, $1C, $1C, $1C
    .byte $FF

pnt1E =*
    .byte $FA, $0B
    .byte $FD, $01
    .byte $1E, $1E, $1E, $1E
    .byte $1E, $1E, $1E, $1E
    .byte $1E, $1E, $1E, $1E
    .byte $1E, $1E, $1E, $1E
    .byte $1E, $1E, $1E, $1E
    .byte $1E, $1E, $1E, $1E
    .byte $1E, $1E, $1E, $1E
    .byte $1E, $1E, $1E, $1E
    .byte $FF

pnt1F =*
    .byte $FA, $09
    .byte $FD, $3F
    .byte $23, $1B, $1C, $1E

```



```

.byte $FF

pnt20 =*
.byte $FA, $01
.byte $FD, $7F
.byte $17, $1F
.byte $FF

.byte $21, $26
.byte $FD, $11
.byte $28
.byte $FF

pnt21 =*
.byte $FA, $15
.byte $FD, $01
.byte $1F, $1F
.byte $FD, $03
.byte $1F
.byte $FA, $0F
.byte $FD, $01
.byte $2E, $27
.byte $FA, $15
.byte $1F
.byte $FD, $03
.byte $1F
.byte $FD, $01
.byte $1F
.byte $FD, $03
.byte $1F
.byte $FD, $01
.byte $FA, $0F
.byte $2F
.byte $FA, $15
.byte $1A, $1D, $1F
.byte $FF

pnt22 =*
.byte $FA, $09
.byte $FD, $01
.byte $13, $13
.byte $FD, $03
.byte $13
.byte $FD, $01
.byte $FA, $00
.byte $2E, $27
.byte $FA, $09
.byte $13
.byte $FD, $03
.byte $13
.byte $FD, $03
.byte $13
.byte $FD, $01
.byte $13, $10, $11, $13
.byte $FF

pnt23 =*
.byte $FA, $09
.byte $FD, $01
.byte $17, $17
.byte $FD, $03
.byte $17
.byte $FD, $01
.byte $FA, $00
.byte $2E, $27
.byte $FA, $09
.byte $17
.byte $FD, $03
.byte $17
.byte $FD, $01
.byte $17
.byte $FD, $03
.byte $17
.byte $FD, $01
.byte $17, $12, $15, $17
.byte $FF

pnt24 =*
.byte $FA, $15
.byte $FD, $01
.byte $23, $23
.byte $FD, $03
.byte $23
.byte $FA, $0F
.byte $FD, $01
.byte $2E, $27
.byte $FA, $15
.byte $23

```

```

        .byte $FD, $03
        .byte $23
        .byte $FD, $01
        .byte $23
        .byte $FD, $03
        .byte $23
        .byte $FD, $01
        .byte $FA, $0F
        .byte $2F
        .byte $FA, $15
        .byte $1E, $21, $23
        .byte $FF

pnt25 =*
        .byte $FA, $0A
        .byte $FD, $39
        .byte $47
        .byte $FD, $01
        .byte $46, $45, $44
        .byte $FD, $39
        .byte $43
        .byte $FD, $01
        .byte $44, $45, $46
        .byte $FF

pnt26 =*
        .byte $FA, $12
        .byte $FD, $3F
        .byte $3B, $43, $42, $3E
        .byte $3B, $37, $36, $2F
        .byte $FF

pnt27 =*
        .byte $FA, $0C
        .byte $FD, $01
        .byte $31, $3D, $49, $3D
        .byte $31, $3D, $49, $3D
        .byte $FF

pnt28 =*
        .byte $FA, $01
        .byte $FD, $7F
        .byte $17, $00, $00, $00
        .byte $FF

pnt29 =*
        .byte $FA, $01
        .byte $FD, $7F
        .byte $23, $00, $00, $00
        .byte $FF

;=====
; arpeggio pointers table
;=====
pointerLo:
        .byte <TableArp0

r157B:
pointerHi:
        .byte >TableArp0

TableArp0:
        .byte $00, $0C, $18

;pointers to the patterns

;low pointers
patptl =*
        .byte <pnt00
        .byte <pnt01
        .byte <pnt02
        .byte <pnt03
        .byte <pnt04
        .byte <pnt05
        .byte <pnt06
        .byte <pnt07
        .byte <pnt08
        .byte <pnt09
        .byte <pnt0A
        .byte <pnt0B
        .byte <pnt0C
        .byte <pnt0D
        .byte <pnt0E
        .byte <pnt0F
        .byte <pnt10
        .byte <pnt11
        .byte <pnt12
        .byte <pnt13
        .byte <pnt14

```

Here goes the definition
of the arpeggio

```

    .byte <pnt15
    .byte <pnt16
    .byte <pnt17
    .byte <pnt18
    .byte <pnt19
    .byte <pnt1A
    .byte <pnt1B
    .byte <pnt1C
    .byte <pnt1D
    .byte <pnt1E
    .byte <pnt1F
    .byte <pnt20
    .byte <pnt21
    .byte <pnt22
    .byte <pnt23
    .byte <pnt24
    .byte <pnt25
    .byte <pnt26
    .byte <pnt27
    .byte <pnt28
    .byte <pnt29

;high pointers

patpth =*
    .byte >pnt00
    .byte >pnt01
    .byte >pnt02
    .byte >pnt03
    .byte >pnt04
    .byte >pnt05
    .byte >pnt06
    .byte >pnt07
    .byte >pnt08
    .byte >pnt09
    .byte >pnt0A
    .byte >pnt0B
    .byte >pnt0C
    .byte >pnt0D
    .byte >pnt0E
    .byte >pnt0F
    .byte >pnt10
    .byte >pnt11
    .byte >pnt12
    .byte >pnt13
    .byte >pnt14
    .byte >pnt15
    .byte >pnt16
    .byte >pnt17
    .byte >pnt18
    .byte >pnt19
    .byte >pnt1A
    .byte >pnt1B
    .byte >pnt1C
    .byte >pnt1D
    .byte >pnt1E
    .byte >pnt1F
    .byte >pnt20
    .byte >pnt21
    .byte >pnt22
    .byte >pnt23
    .byte >pnt24
    .byte >pnt25
    .byte >pnt26
    .byte >pnt27
    .byte >pnt28
    .byte >pnt29

songs =*
ltr1:
    .byte $00
    .byte <drillerMainTr1

htr1:
    .byte $00
    .byte >drillerMainTr1

ltr2:
    .byte $00
    .byte <drillerMainTr2

htr2:
    .byte $00
    .byte >drillerMainTr2

ltr3:
    .byte $00
    .byte <drillerMainTr3

htr3:

```

The tables of songs
with pointers to the
tracks for each voices

```

.byte $00
.byte >drillerMainTr3

.byte $00

initsongs =*
    lda #$01          ; first song
    sta songsNumber
    rts

```

Conclusion

May be now you know better how Matt had created a so beautiful sound.

However, you should experiment with the source if you want to understand better how the engine works.

The last thing is rasterlines: the minimum used is of 10 (used when creating the sound effects), while when reading new notes/patterns, the maximum grown until 34!

Finally, as far as I know, the Matt' engine was used in the past even by Tomas Danko and Jon Wells for a total of 11 tunes.

OSDin 2 end