hinweis der einfachheit halber wird auf gross/kleinschreibung verzichtet, wir werden dich mit "du" ansprechen. wenn im text eine taste auf der tastatur gemeint ist, so wird diese mit grün hinterlegt, dinge die im goattracker zu lesen sind werden mit cyan gekennzeichnet. hinweise die gelb gekennzeichnet sind sollte man etwas genauer nehmen, da sie auf essentielle dinge aufmerksam machen deren ignorierung viele stunden ärger oder unnötige grübelei nach sich ziehen kann. die musikalische note B ist im deutschen sprachraum als H bekannt, trotzdem wird in dieser anleitung, so wie in goattracker, das B verwendet.

inhalt

hinweis in der anleitung werden folgende englische begriffe verwendet,
so wie sie teilweise auch im programm zu sehen sind bzw. allgemeingültig sind:

table > tabelle, wir verwenden das wort 'table' so wie es im tracker zu sehen ist
waveform > wellenform
register > eine speicherstelle die einen wert speichert (ein byte)
speed > geschwindigkeit
command > befehl, kommando
bit > kunstwort aus BInary digiT, kleinste informationseinheit, 0 oder 1
byte > kunstwort aus beissen > bite, aus 8 kleinen bissen [bit] wird ein byte
nybble > kunstwort aus knabbern > nibble, nur ein kleiner happen, 4 bit
hex > hexa (sechs) dezi (zehn) > hexadezimalsystem, basis 16
signed (hex) **value** > hexadezimalzahl in vorzeichenbehafteter darstellung
low > kein signal, nicht geladen, 0
high > geladen, 1
offset > versatz (offset $01 = $01 dazuaddieren/subtrahieren)
attack > anschlag, anklingen
decay > abfall
sustain > halten
release > ausklingen
gate > tür, gatter - z.b. beim gatebit - das gatter wird quasi aufgemacht (das gate bit gesetzt), der klang ist hörbar.
noise > rauschen
frame > rahmen, 1 frame = eine schwingung des normalen 50hz bildschirmupdates
tune > viele sprechen von "tunes" wenn von .sid songs die rede ist
pitch > tonhöhe
channel, voice > stimme, oszillator - einzelne stimme, wird synonym verwendet
space > leertaste
trigger > auslösen
fixed > fixiert, nicht änderbar
grid > pattern grid - gitter, die wahrnehmung des pattern editors als raster
arpeggio > eine schnelle tonfolge als alternative zum akkord
oldschool > musiker der alten schule - wie hubbard, galway usw.
pulse > eine rechteck wellenform, wir verwenden nur den namen "pulse" im tutorial
PWM > pulsweitenmodulation, im deutschen auch "pulsbreitenmodulation" oder z.b. im c64 handbuch "tastverhältnis" genannt
triangle > englische bezeichnung für 'dreieck' wellenform
saw > englische bezeichnung für 'sägezahn' wellenform
synthesizer > ein gerät um klang künstlich zu erzeugen, zb. moog
cutoff > die "abschneide" frequenz eines filters, auch grenzfrequenz genannt - die frequenz bei der ein filter am klang "zu greifen" beginnt.
resonance > eine rückkopplung des filters auf sich selbst
pattern > eine zusammengefasste abfolge von tönen und befehlen in einer art anweisungsliste wird pattern genannt.
loop > wenn etwas "im kreis" bzw. wiederholt abgespielt wird
sweep > der verlauf eines wertes, zb. ein filter (cutoff) sweep

kapitel 0 - binäres und des hexadezimales zahlensystem

wenn du bereits binär und hex "denken" kannst, und du weisst was ein vorzeichenbehafteter hex wert in zweierkomplementdarstellung ist, dann kannst du bei kapitel 1 weiterlesen. wir sind der meinung das das wissen aus kapitel 0 ein *grundstein* ist ohne den der soundchip kaum sinnvoll nutzbar ist. die angaben in kapitel 0 beziehen sich explizit auf die im commodore 64 vorfindbare 6502 mikroprozessor & little-endian architektur und können nicht einfach allgemeingültig für andere plattformen übernommen werden, ebenso bezieht sich die erklärung der vorzeichenbehafteten ("signed") wertigkeiten nur auf die zweierkomplement darstellung wie sie im goattracker zur anwendung kommt.

kapitel 0.1 - BIT & nybble

in unserem alltagseben haben die meisten von uns nur mit dezimalzahlen zu tun. rechner können mit diesen zahlen nichts anfangen, im rechner gibts nur "ladung" (1) oder "keine ladung" (0), darauf basiert das binäre zahlensystem mit dem alle rechner arbeiten. ein einzelner "ladung"/"keine ladung" wert wird BIT genannt. vier dieser werte zusammengenommen stellen ein "nybble" dar. binärzahlen sind mit einem % zeichen davor gekennzeichnet. der niedrigste wert den ein nybble haben kann ist %0000, der höchste ist %1111. wenn wir jetzt alle möglichen 0 und 1 kombinationen durchprobieren erkennen wir des ein nybble 16 verschiedene werte annehmen kann. das binäre system ist auf verdopplung des grundwertes aufgebaut, das niedrigste bit hat einen wert von 1. demnach sind die dezimalen wertigkeiten der bits von LSB nach MSB: 1 2 4 8 (und 16 32 64 128 - wenn es zwei nybble sind bzw. ein byte ist) ein bit das eingeschaltet, geladen, auf "1" gestellt ist heisst, dass wir den dezimalen wert des bits zur summe dazuzählen müssen, um den engültigen wert des nybbles zu berechnen.

das niedrigwertigste bit wird als LSB (least significant bit), das bit das den wert am wenigsten beeinflusst, bezeichnet. umgekehrt ist das MSB (most significant bit) das bit das den wert am meisten beeinflusst.

wenn wir das jetzt aufschlüsseln:

```
                  MSB             LSB
bit nummer          3    2    1    0
dezimalwert         8    4    2    1
                    :    :    :    :
                    :    :    :    :
beispiel            :    :    :    :
binärwert           %1   0    1    0  =  dezimal 10 (8 + 2)
```

kapitel 0.2 - HEX

hexadezimal bedeutet das das zahlensystem eine basis von 16 hat (wo unser "normales" zahlensystem eine basis von 10 hat). das war die naheliegendste möglichkeit, da ein nybble einen wert von dezimal 0-15 halten kann. da das dezimalsystem nur 10 "symbole" für zahlen hat wurden die buchstaben von A bis F dazugenommen. nachdem ein hex wert über 9 hinaus geht wird bei A weitergezählt, bis F. Hex zahlen werden mit einem $ davor dargestellt (verwechseln von hex und dezimal stiftet viel verwirrung!).

eine einfache umrechnungstabelle:

dez	bin	hex
0	%0000	$0
1	%0001	$1
2	%0010	$2
3	%0011	$3
4	%0100	$4
5	%0101	$5
6	%0110	$6
7	%0111	$7
8	%1000	$8
9	%1001	$9
10	%1010	$A
11	%1011	$B
12	%1100	$C
13	%1101	$D
14	%1110	$E
15	%1111	$F

kapitel 0.3 - byte & % $ umrechnung

ein byte besteht aus 8bits und ist daher doppelt so breit wie ein nybble. Das nybble mit dem LSB das "low nybble" ist, während das nybble mit dem MSB das "high nybble" genannt wird. das MSB hat die wertigkeit 128 dezimal (oder $80 hex). spätestens jetzt bist du endgültig verwirrt. aber lies weiter, es lichtet sich. der grösste wert den ein byte haben kann ist 255 dezimal oder $FF hex, %1111 1111 in binärdarstellung.

der aufbau eines bytes:

	MSB						LSB
bit nummer	7	6	5	4	3	2	1	0
dez wert	128	64	32	16	8	4	2	1
hex wert	$80	$40	$20	$10	$08	$04	$02	$01

der trick bei einem byte ist, das es behandelt werden kann wie zwei nybbles für einfache bin>hex (und retour) umrechnungen.

%10110110
teilt sich zu

%1011 %0110

dann einzeln betrachtet
%1011 = $B
und
%0110 = $6

ergibt:
%1011 0110 = $B $6 = $B6

in die andere richtung:
$E4
teilt sich in
$E = %1110
und
$4 = %0100
ergibt:
$E4 = %1110 0100

bytes als nybbles addieren geht auch einfach:
```
  $10  %0001 0000
+ $40  %0100 0000
-----------------
  $50  %0101 0000

  $20  %0010 0000
+ $40  %0100 0000
-----------------
  $60  %0110 0000
```

obwohl es auch kompliziert werden kann wenn wir zu höheren werten addieren, in dem fall ist es einfacher das ganze byte zu sehen anstatt den nybbles:

```
  $7F  %0111 1111
+ $01  %0000 0001
-----------------
  $80  %1000 0000
```

$7F - low nybble ist schon am höchsten wert ($F). wäre das dezimal system, wäre der höchste wert dementsprechend 9. addiert man dec 79+1 erhält man 80. in hex ist es das selbe, wir erhöhen das high nybble um einen wert und setzen das low nybble wieder auf $0. $7F + $01 = $80 man muss nur den dreh rauskriegen das nach $9 die "zahl" $A kommt.

tipp: natürlich kann man das ganze mit dem windows taschenrechner auch umrechnen, ansicht > programmierer.

kapitel 0.4 - vorzeichenbehaftete bytes

ein vorzeichenbehaftetes ("signed") byte nennt man so weil das MSB des bytes anzeigt ob der wert des bytes positiv oder negativ ist. dadurch bleiben nur mehr 7 bit übrig um einen wert darzustellen, daher reicht ein vorzeichenbehaftetes byte von dez -128 bis dez +127. in der darstellung werden zusätzlich die bits invertiert (umgedreht) im fall das das byte einen negativen wert hat. %0000 0000 ist dez 0, ein vorzeichenbehaftetes byte mit %1111 1111 ist dez -1. zusätzlich wird bei einem negativen wert ein offset (versatz) von $01 addiert nachdem die bits invertiert wurden.

umrechnung von positiv nach negativ:

```
$3E = %0011 1110 (=62)
invertiert = %1100 0001
+ offset     %0000 0001
----------------------
ergebnis     %1100 0010 = $C2 (= -62)
```

```
und hier der grund warum der offset benötigt wird,
umrechnung $01 postiv nach negativ:

$01 = %0000 0001
invertiert = %1111 1110
+ offset     %0000 0001
---------------------
ergebnis     %1111 1111 = $FF (= -1)

vorzeichenbehaftete byte werte reichen also von:

$01 bis $7F = dez 1 bis dez 127
$FF bis $80 = dez -01 bis dez -128

positive reichweite:
$01=$+01
$02=$+02
$03=$+03
.
.
.
$7D=+$7D
$7E=+$7E
$7F=+$7F

negative reichweite:
$FF=$-01
$FE=$-02
$FD=$-03
.
.
.
$82=$-7E
$81=$-7F
$80=$-80
```

kapitel 1 - grundlagen & einstieg

kapitel 1.1 - grundlegende definierung des soundchips

der c64 soundchip (6581/8580) ist tatsächlich ein kleiner dreistimmiger synthesizer der eine breite palette von elektronischen und sogar teilweise organischen sounds von sich geben kann. ähnlich einer orgel in einer kirche, hat auch der soundchip 27 register die man stellen kann um den sound zu beeinflussen. ein register ist in unserem fall nichts anderes als eine stelle im speicher die den wert von einem byte speichern kann. der soundchip hat drei oszillatoren, oder für den musiker: drei stimmen (vergleich: ein mensch nur eine stimme). zusätzlich gibt es noch ein analoges filter durch den die drei stimmen jeweils optional geschickt werden können.

klangerzeugung im sid wird primär durch folgende vier faktoren bestimmt:

1. tonhöhe - frequenz (pitch) - jede sid stimme hat zwei register um die frequenz einzustellen, also einen 16 bit wert. wir müssen uns darum aber nicht kümmern, sobald man im goattracker eine note eingibt werden automatisch die richtigen werte für die jeweilige frequenz in die register der jeweiligen stimme geschrieben.

2. lautstärke - amplitude - bei synthesizern und auch beim sid wird das konzept der ADSR hüllkurve angewandt. in synthesizern wird mit der ADSR hüllkurve ein spannungsgesteuerter

verstärker automatisiert, beim sid ist das alles im chip integriert und wir müssen nur die ADSR werte einstellen um den lautstärkenverlauf zu bestimmen.

3. wellenform - ein klang besteht aus druckveränderungen in der luft, durch schwingung. der sid ist in der lage verschiedene schwingungen bzw. wellenformen zu erzeugen.

4. filter - durch das analoge filter können verschiedene anteile der wellenform subtrahiert oder auch hervorgehoben werden.

in einem satz: die oszillatoren im sid erzeugen wellenformen in einer gewissen lautstärke, die mit einer definierten frequenz schwingen, diese werden optional durch einen filter verändert.

kapitel 1.2 - ADSR

bedeutet:
attack zeit - die zeit bis zur vollen lautsärke, vor dem decay
decay zeit - zeit bis der sustain pegel erreicht ist
sustain pegel - klang bleibt im sustain (sustain ist ein pegel, keine zeit!)
release zeit - release, sound klingt aus

anders erklärt:
attack - anschlag - du drückst eine taste am klavier,
der hammer schlägt an die saite,
die saite ertönt in voller lautstärke
decay - abfall - die lautstärke wird geringer
sustain - halten - die saite schwingt weiter mit normaler lautstärke
(taste noch immer gedrückt)
release - ausklingen - loslassen der taste, die saite klingt aus und verstummt

jeder dieser 4 faktoren wird durch einen nybble wert dargestellt den man im instrumenten editor einstellen kann. $0 = kurz, $F = lang. im falle des sustains - $F = laut. achtung $F bei attack dauert 8 sekunden, $F bei decay/release dauert 24 sekunden. wenn bit 0 des wellenform registers gesetzt wird, dann wird dieser ADSR verlauf gestartet. man nennt diesen teil eines synthesizers auch hüllkurvengenerator.

dieses bit 0 wird auch "gate bit" genannt, da der klang erst hörbar wird sobald das gate bit gesetzt ist. also im verlauf: wenn diese bit auf 1 gesetzt wird (=gate on) wird der attack/decay zyklus gestartet, nachdem die decay zeit vorüber ist wird der klang mit dem sustain level gehalten. wenn das wellenform register bit 0 (eben das gate bit) wieder auf 0 gesetzt wird folgt der release und der klang verstummt.

diagramm für ein instrument mit schnellem attack, schnellem decay und langem release:

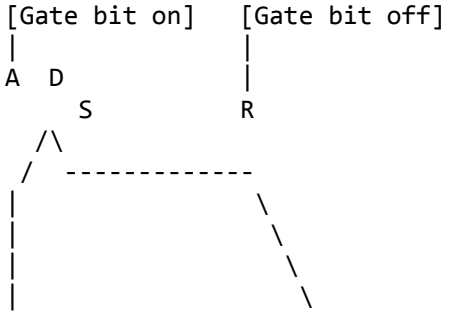
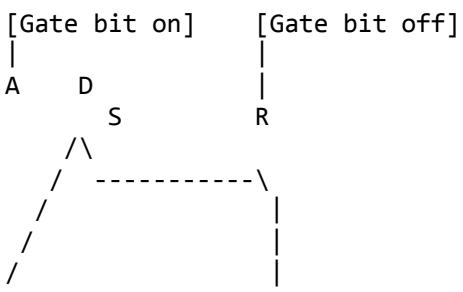


diagramm für ein instrument mit langsamen attack, schnellem decay und kurzem release:



kapitel 1.3 - wellenform - wavetable

im goattracker instumenten editor gibt der $ wert neben [Wavetable Pos] an ab welcher position im wavetable die wellenform bytes stehen, die frame für frame in die sid register geschrieben werden. 1 frame = 1 bildschirmupdate = 1 wert wird aus dem table gelesen und in das register geschrieben. wenn leute von 2x oder 4x tunes reden, dann meinen sie damit das die abspielroutine mehrmals pro bildschirmupdate aufgerufen wird, dadurch können viel mehr wellenformen und parameter in der selben zeit in die register geschrieben werden, dadurch bekommt man mehr klangliche möglichkeiten. für den anfang reicht aber 1x.

der [WAVE TBL] besteht aus zwei spalten, die erste besteht aus den wellenform register werten, die rechte gibt relativen/absoluten pitch offset an, darüber werden wir später sprechen.

hinweis verwende den [TABULATOR] um zwischen pattern/orderlist/instrument/wavetable/songname zu wechseln. für direktwechsel zwischen instrument und wavetable, verwende [F7]. du wirst effektiver und schneller komponieren wenn du deine maus am besten gar nicht verwendest.

wie wir bereits gehört haben steuert bit 0 das ADSR gate. was machen die anderen 7 bits des wellenform registers?

	MSB						LSB
bit	7	6	5	4	3	2	1	0
hex	$80	$40	$20	$10	$8	$4	$2	$1
	rauschen	pulse	sägezahn	dreieck	test	ring	sync	ADSR

Bit 7 wählt die pseudo generierte "farbiges rauschen" wellenform, praktisch für hihat und snaredrum sounds. die höhe des rauschens ist abhängig davon welche note man damit spielt, trotzdem kann man "rauschen" nur bedingt für melodien verwenden.

Bit 6 pulse wellenform, sieht ca. so aus: __---__---__--- achtung man muss die pulsweite einstellen, sonst hört man diesen klang nicht (siehe kapitel 2).

Bit 5 sägezahn wellenform, sieht so aus wie sie heisst: /|/|/|.

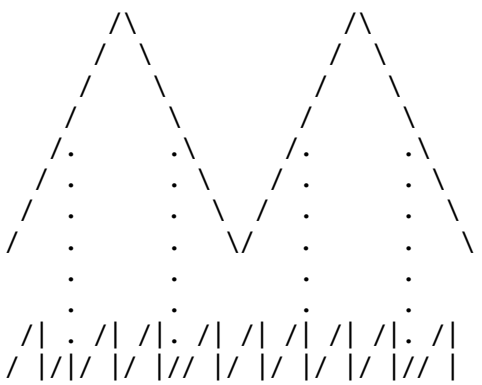
Bit 4 dreieck wellenform, sieht ebenso so aus wie sie heisst: /\/\/. hat einen sehr weichen klang, gut für flöten klänge.

Bit 3 "test" oder "reset" bit. C64 oszillatoren hängen sich auf wenn sie zusammen mit rauschen selektiert werden. mit diesem bit kann man den oszillator rückstellen, und auch die wellenform "neu starten". über das oszillator aufhäng problem muss man sich keine sorgen machen, goattracker regelt das von selbst mit der hardrestart routine (die auch dafür sorgt das es keine probleme mit der adsr sektion gibt).

Bit 2 "ringmodulation" - wenn dieses bit auf 1 gesetzt wird, wird der oszillator (der klang muss dreieck enthalten) mit der frequenz des vorherigen oszillators ringmoduliert, egal welche wellenform mit dem vorherigen oszillator verwendet wird. vorheriger oszillator bei goattracker heisst: der oszillator bzw das pattern links neben der modulierten stimme (die drei patterns im editor sind den drei sid stimmen zugewiesen). wenn man ringmod auf stimme 1 anwendet ist stimme 3 die vorherige. der effekt ist leichter gehört als beschrieben, es klingt sehr metallisch und kann für FM ähnliche glockenklänge verwendet werden. mitunter ist der effekt schwierig in einem musikalischen rahmen zu verwenden, mit etwas geduld in der notenauswahl für die beteiligten oszillatoren kann man es aber nutzen.

Bit 1 "sync" - wenn dieses bit auf 1 gesetzt wird, wird die fundamentale frequenz des oszillators mit der frequenz des vorherigen oszillators "hart" synchronisiert. technisch gesehen wird die wellenform die der oszillator gerade abspielt neu gestartet wenn die amplitude des vorherigen oszillators in der mitte den nulldurchgang durchläuft.

diagramm für sync - oszillator 1 spielt eine dreieck welle, oszillator 2 einen sägezahn, der einen höheren pitch (tonhöhe) als oszillator 1 hat:



Bit 0 "ADSR"/ gate - bit um den lautstärkenverlauf zu steuern

durch das wissen das wir bis jetzt erlangt haben, können wir unseren ersten eigenen sound erstellen, öffne goattracker und springe in den instrument editor [F7] und stelle die werte ein:

edit [Attack/Decay 55]
und [Sustain/Release 55]
edit [Wavetable Pos 01]

springe zu WAVE TBL mit [F7], edit
[01:21 00] (wählt sägezahn wellenform und setzt gatebit)
[02:20 00] (sägezahn wellenform immer noch gesetzt, gatebit off)
[03:FF 00] (FF is ein sprungbefehl zum wavetable programmieren.
FF 00 heisst - stoppe die wavetable ausführung)

drücke [SPACE] um eine "test note" mit diesem instrument zu spielen (verwende * und / für oktavlage)

da das gatebit nur für einen frame "on" ist, kann man den attack nicht
wirklich hören. ändern wir die wavetable position 02 zu:

[02:FF 01] (springe zu 01)

drücke [SPACE] wenn wir in den instrument editor zurückspringen, können wir attack
und decay sowie sustain verändern und mit [SPACE] die note nocheinmal triggern um die veränderung zu hören.

beende den sound durch drücken von [F4] oder lösche das gatebit.

wenn du die letzten drei worte im vorherigen satz nicht zu 100% verstanden hast solltest du nocheinmal weiter vorne beginnen zu lesen, es wird ab jetzt immer komplexer und ohne grundlagen dann wird vieles unklar bleiben.

man kann auch total abgefahrene sounds erstellen:

edit:
[01:11 00]
[02:21 00]
[03:51 00]
[04:FF 01]

hinweis du kannst die [Einfg] und [Entf] tasten verwenden um werte nach unten oder oben zu verschieben. wenn du den cursor auf die 01 position bewegst und [Einfg] drückst, siehst du auch gleich das goattracker automatisch die "Wavetable Pos" mitverändert, je nachdem wo du den anfang des sounds hinverschiebst. dadurch muss man nicht nachträglich alles editieren wenn man mitten im table einen sound verändern will.

kapitel 1.4 - arpeggios

wie bereits erwähnt wird die rechte spalte im WAVE TBL für pitch offset (versatz) verwendet. der versatz kann relativ (zum notenwert addiert/davon subtrahiert) oder absolut (fix definierte tonhöhe die sich nicht ändert) sein.

$00 - $5F positive relative noten
$60 - $7F negative relative noten
$80 - tonhöhe bleibt unverändert
$81 - $DF - absolute noten C#0 - B-7

arpeggios - da der sid chip nur begrenzt polyphon ist wurde das konzept der arpeggios angewandt. ein arpeggio ist eine folge von noten (eine tonleiter oder ein zerlegter akkord) die mit schneller geschwindigkeit abgespielt wird. ein sehr charakteristischer c64 sound. wenn wir z.b. den für e-gitarren typischen "powerchord" als arpeggio nachbauen, müssen wir nur die bünde von der ersten bis zur zweiten note zählen (den intervall in halbtonschritten). die dritte note im powerchord ist einfach die oktave der ersten. der powerchord ist sehr praktisch, da er nur dreistimmig und weder dur noch moll ist, auch ist er in allen tonlagen spielbar und somit ein universelles arpeggio.

im wavetable sieht das so aus:
[01:21 00]
[02:21 07] (+7 halbtonschritte =eine quinte)
[03:21 0C] (+12 halbtonschritte =eine oktave)
[04:FF 01] (endlosschleife - jump 01)

geh mit [F5] in den pattern editor und drücke [SPACE] um zum [JAM MODE] (wird links unten angezeigt) zu wechseln und das arpeggio auf der keyboard klaviatur zu spielen.

hinweis markieren der "schwarzen" klaviertasten am qwertz keyboard vereinfacht das komponieren anfangs um einiges. es gibt auch zwei verschiedene tastaturlayouts: fasttracker und DMC. dieses tutorial bezieht sich durchwegs auf die fasttracker belegung (die als default voreingestellt ist).

hinweis [SPACE] schaltet zwischen JAM und EDIT um, so kann man melodien ausprobieren ohne das derzeitige pattern zu verunstalten.

hinweis wenn deine rechnerplattform ein mac ist, kannst du auch ein midi keyboard anschliessen für komfortable noteneingabe.

wenn du dein eigenes arpeggio entwickeln willst, such dir ein paar noten heraus die gut miteinander klingen bzw. drücke ein paar tasten auf der keyboard klaviatur und merke dir diejenigen die gut zusammenpassen. wenn man jetzt die tiefste note nimmt kann man die tasten dazwischen abzählen:

beispielsweise:
tiefste note wäre [C] (E-3)
zweite note sollte sein [N] (A-3)
jetzt kann man die tasten nach oben hin abzählen von [C] bis [N], [C] als grundton wird nicht mitgezählt, die zweite note [N] wird aber sehr wohl mitgezählt.

das ist dann [V] [G] [B] [H] [N] = 5 = $05 relativer offset

deshalb ist es auch hilfreich die halbtonschritte (schwarze tasten) am keyboard zu markieren, beim abzählen weiss man nicht immer gleich wo ein halbtonschritt liegt.

zur einfacheren -von hand- konvertierung von gitarrentabulatur nach sid arpeggios gibt es auch das simple tool "tabtoarp", auf csdb.dk zu finden.

kapitel 1.5 - fixed pitch sounds - drums

werte von $81 (C#0) bis $DF (B-7) in der rechten WAVE TBL spalte definieren das der sound mit einer fix definierten tonhöhe abgespielt wert, egal welchen wert man im pattern editor eingibt. das ist praktisch für schlagzeugsounds oder effekte die immer gleich klingen sollen, man kann dann viel schneller zwischen diesen sounds umschalten und rythmische muster eingeben ohne auf die eigentliche eingegebene note acht zu geben. nur die position des sounds im pattern grid (gitter) ist dann wichtig.

schnelles beispiel für einen oldschool drumsound:

edit instrument
[Attack/Decay 32]
[Sustain/Release F9]
[Wavetable Pos 01]

edit WAVE TBL
[01:51 90] (kombiniert pulse & dreieck ($40 + $10 = $50), gate bit setzen
($50 +$01=$51) spiele note mit fixed pitch $90)
[02:21 A0] (wellenform sägezahn, gate immer noch gesetzt, fixed pitch $A0)
[03:80 DF] (wellenform rauschen, gatebit 0, ausklingen mit fixed pitch $DF)
[04:FF 00] (beende wavetable ausführung, sound klingt immer noch aus)
gehe in den pattern editor und teste den sound auf verschiedenen tasten - es wird immer gleich klingen.

hinweis mit goattracker wird ein .pdf von ravenspiral installiert, dort gibt es eine gut gemachte übersichtstabelle über die notenwerte.

kapitel 1.6 - 6581 oder 8580?

gemischte wellenformen am 6581
vom design her sollte es möglich sein pulse, sägezahn und dreieck frei miteinander zu kombinieren, durch ein logisches UND sobald das respektive bit auf 1 gesetzt wird. die rauschen wellenform kann nicht mit den anderen kombiniert werden. durch diverse fehler im 6581 ist es nicht vollständig möglich alle wellenformen zu mischen. pulse und dreieck ($50) ergibt ein gut hörbares egebnis während dreieck und sägezahn ($30) fast nicht hörbar sind. interessant wird es wenn man eine breite pulse wellenform dazugibt. generell sollte man aber versuchen nur die gemischten wellenformen zu verwenden die klar hörbar sind, es ist auch möglich das gewisse mischungen auf einem chip besser klingen als auf anderen.

gemischte wellenformen am 8580
am 8580 kann man mit allen wellenform mischungen klar hörbare resultate erzielen, insgesamt hat man also sieben verschiedene wellenformen. noch mehr variation ist möglich wenn man die pulsweite bei mischung mit pulse verändert (mehr dazu später).

filter beim 6581
ein grosser unterschied zwischen beiden chips ist auch der filter. generell kann gesagt werden das jeder 6581 anders klingt, egal welche revision und egal welcher date code darauf steht. die filterkurve im 6581 ist nicht wirklich konform von chip zu chip, teilweise können damit sehr organische sounds erzeugen und auch sowas ähnliches wie eine verzerrte gitarre hinbekommen (im titellied von mechanicus hört man das sehr gut). wenn man im goattracker mit dem 6581 filter arbeiten will muss man sehr genau wissen was man tut, am echten 6581 wird es definitiv anders klingen. auch ist oft die untere und obere grenze des filters ein wenig verschoben. wenn du einen c64 mit 6581 hast kannst du auch mit sid audit die grenzen des filters testen um einen anhaltspunkt zu haben (mehr zum filter später).

filter beim 8580
der filter im 8580 ist hingegen sehr verlässlich und von chip zu chip gibt es nur wenig unterschiede, auch der klang ist etwas "elektronischer" als der filter im 6581. in der c64 demoszene hat sich der 8580 ziemlich durchgesetzt weil der chip mehr möglichkeiten bietet und die ergebnisse viel reproduzierbarer sind. trotzdem gibt es noch einige die den 6581 wegen seiner charakteristischen bugs bzw. features bevorzugen. trivia: oldschool legende ben daglish ging gleich einen anderen weg - er verwendete gleich gar keinen filter weil er von anfang an ausschliessen wollte das seine tunes überall anders klingen.

kapitel 2 - pulse wellenform - automatisierung, modulation

kapitel 2.1 - einleitung in die PWM thematik - praktische beispiele

die pulse wellenform ($40) ist etwas spezieller da wir das tastverhältnis frei bestimmen können, das heißt wir können definieren wie lange der pulse high und danach low bleibt (manchmal wird das PWM genannt - pulsweitenmodulation).

der soundchip hat sogar zwei register für jede stimme um dieses verhältnis zu definieren, obwohl ein register zu einem nybble reduziert wurde, somit haben wir einen dezimalbereich von 0-4095, oder bessergesagt $000 - $FFF.

trotzdem werden sich die meisten jetzt noch immer nichts darunter vorstellen können, obwhohl PWM auch ein sehr charakteristischer c64 sound ist.

ein komplett symmetrischer pulse sieht folgendermassen aus:
```
high  ----    ----    ----    ----
low       ----    ----    ----    ----
```

wenn wir jetzt die puls***weite*** verändern würde es so aussehen:
```
high  ------  ------  ------  ------
low         --      --      --      --
```

klanglich verändern sich die harmonien des sounds, gewisse obertöne werden ausgelöscht während andere verstärkt werden, je nach high zu low verhältnis.

wir editieren einen PWM sound:
editiere die AD SR werte im instrument editor nach belieben gib ein:
[Wavetable Pos 01]
[Pulsetable Pos 01] (genauso wie beim wavetable ist das jetzt der zeiger auf den PULSETBL)

edit WAVE TBL
[01:41 00] (pulse wellenform einstellen und gate on)
[02:FF 00]

wie bereits erwähnt hören wir nichts, weil im register immer noch $000 steht. also müssen wir ein paar werte im PULSETBL eingeben (der PULSETBL wir genauso wie der wavetable frameweise aus- geführt)

edit
[01:98 00]
[02:FF 00]

drücke [SPACE] - was du jetzt hörst ist eine symmetrische pulse wellenform

01:98 00 bedeutet: ein wert im pulsetable wo das am meisten links stehende nybble im bereich von $8 bis $F ist (bei uns $9) definiert die aktuelle pulsweite. man könnte auch [01:F8 00] eingeben und den selben effekt erlangen. das zweite nybble ($8) ist das high nybble (MSB) zum einstellen der pulsweite, also wird dadurch der klang am meisten beeinflusst. die zwei rechten nybbles sind das low byte der pulsweite und werden für feinabstimmung verwendet.

edit PULSETBL
[01:9X 00] - versuche verschiedene werte für X einzugeben, drücke [SPACE] um den sound klingen zu lassen. vielleicht hast du schon gemerkt das der klang dünner wird desto näher du zu $000 oder $FFF kommst. das liegt daran das eine pulsweite mit $000 oder $FFF so weit ist das ein konstantes high oder low entsteht, dadurch entsteht keine schwingung und man kann nichts hören.

$000 and $FFF sind genau gegenüberliegend bzw. invertiert:

pulsweite $000
high
low ----------
pulsweite $FFF

high ----------
low
deswegen gibt unser anfangswert $800 eine symmetrische pulse wellenform, weil das genau die mitte zwischen $000 und $FFF ist. von $000 nach $800 ergibt den selben klang wie von $FFF nach $800, obwohl es physikalisch nicht das selbe ist. wenn man mehrere instrumente mit pulse verwendet sollte man versuchen auch die gegenüberliegende seite zu nutzen. obwohl es für die einzelsounds klanglich keinen unterschied macht trägt es zum gesamtklang des tunes bei, man bekommt einen volleren sound.

kapitel 2.2 - PWM automatisierung & modulation - pulsetable

gehe in den PULSETBL und edit
[01:98 00] (setze pulsweite auf $800)
[02:6C 13] (erhöhe pulsweite um $13 für die dauer von $6C frames)
[03:FF 00] (beende pulse programm)

drücke [SPACE] und du hörst einen pulsweiten verlauf von $800 bis $FFF. das konzept dieser automatisierung ist etwas eigenartig für einen musiker, da es eher aus der programmierer ecke kommt. stell dir vor, die PULSETBL automatisierung ist einfach nur eine tabelle mit werten und goattracker (oder später die playroutine) addiert oder subtrahiert diese werte im pulsweiten register. wie geht das?

[02:6C 13] werte in der linken spalte die von $01 bis $7F reichen zeigen an das die pulsweite moduliert bzw. automatisch verändert werden soll. in unserem fall heisst $6C $13 - ADDIERE einen wert von $13 zum pulsweiten register für die dauer von $6C frames. das rechte byte, in der rechten spalte, ist ein vorzeichenbehafteter wert!

wenn wir $6C auf $1F verringern [02:6C 01], dann verändert sich die pulsweite nur langsam, weil nur ein wert von $01 zum register addiert wird.

wenn wir den schritt 02 jetzt verändern:

[02:7F 20] und [SPACE] drücken hören wir das der klang über $FFF hinaus geht (über das konstante high, wo man den klang nicht hört) und wieder von vorne anfängt. das ist deswegen weil auch die register wieder von vorne anfangen. wenn man ein register mit dem wert $FF um $01 erhöht fängt es wieder bei $00 an (für die programmierer: und das carry flag wird gesetzt).

wenn also der wert über $FFF hinaus geht und die pulsetable ausführund nochmal $20 hinzufügt, dann geht es bei $000 weiter so lange bis alle $7F frames abgearbeitet sind und die pulsweite bei ca. $800 zu stehen kommt.

viele sehen den klang wenn die pulsweite bei $FFF zu $000 überschlägt als nicht hörenswert, ideal ist es wenn man also versucht seine pulsprogramme so zu halten das die modulation nicht über diesen punkt hinaus geht.

hinweis $7F ist die maximale dauer für die automation, ab $8x aufwärts ist, wie wir bereits wissen, reserviert für einstellen der pulsweite.

noch ein diagramm zur pulsweite und dem totpunkt:

```
$800 __--__--__--__-- (symmetrisch)
$900
$A00
$B00
$C00 -_---_---_---_-- (3 high/1 low)
$D00
$E00
$FFF ---------------- (stille)
$000 ________________ (stille)
$100
$200
$300
$400 _-___-___-___-__ (1 high/3 low)
$500
$600
$700
$800 __--__--__--__-- (symmetrisch)
```

tipp: probiere den [FF] befehl im pulsetable aus. wenn du [03:FF 02] eingibst wird der sound durch den FF sprungbefehl unendlich lange moduliert. man kann auch die breite in einer serie von $8x befehlen direkt einstellen, man muss nicht automatisieren. oder man kann auch einen schnellen modulationsbefehl und danach einen langsamen eingeben, also mehrere befehle in serie. der pulsweiten-startpunkt für einen befehl ist dann immer dort wo das vorherige kommando die pulsweite stehen hat lassen.

das byte in der rechten spalte, der wert der subtrahiert oder addiert wird, ist wiegesagt ein vorzeichenbehafteter wert. wenn man also bei einer gewissen pulsweite stoppen will und den selben klang nochmal rückwärts haben will muss man einen negativen (vorzeichenbehafteten) wert addieren. sehr eindrucksvoll ist es wenn man zum pulsweitenregister dazuaddiert und vor dem totpukt wieder subtrahiert und diese zwei vorgänge via FF befehlt endlos durchlaufen lässt.

noch ein hinweis natürlich funktioniert das ganze auch umgekehrt, wenn man von $000 subtrahiert beginnt das register wieder bei $FFF.

kapitel 3 – filter

kapitel 3.1 - verwendung des filters

einige leute nennen den sid chip auch einen subtraktiven synthesizer, und das ist er auch in dem sinn das der filter einen teil des gesamtklanges abschneidet bzw. eben filtert. filter beim sid werden mittels des frequenzbereichs in dem sie arbeiten unterschieden, drei verschiedene typen sind es die man frei miteinander kombinieren kann.

lowpass - nur niedrige (Low) frequenzen dürfen durch (pass)
bandpass - nur mittlere frequenzen dürfen durch
highpass - nur hohe frequenzen dürfen durch

zusätzlich dazu kann man das cutoff (grenzfrequenz des filters) register einstellen das eben definiert bei welcher frequenz der filter "greift". also wenn man lowpass einstellt und cutoff auf den wert $20, dann werden alle frequenzen unter $30 durchgelassen, der klang wird also sehr dumpf klingen. genaue angaben wie man diese frequenzen nach hex berechnet finden sich im handbuch, in der praxis probiert man aber ein paar werte und entscheidet mit dem ohr was gut klingt.
bandpass lässt nur frequenzen durch die im bereich des cutoff werts in der mitte des frequenzspektrums liegen, also keine bässe oder hohen frequenzen. manche nennen den bandpass auch "MIDpass"

highpass lässt alle frequenzen über dem cutoff wert durch, also können keine tiefen oder mittig spektralen sounds gehört werden.

des weiteren kann man die resonanz des filters einstellen, höhere resonanz bedeutet die frequenzen im cutoff bereich werden angehoben, der filtersound wirkt kräftiger. am frequenzspektrum kann man sich es so vorstellen wie das dach eines zirkuszeltes das im wirkungsbereich des filters mittig herausgezogen wird, je nach einstellung können dadurch erwünschte oder unerwünschte verzerrungen entstehen.

kapitel 3.2 - filtertable

durch das bisher angeeignete wissen erstellen wir ein instrument mit viel sustatin, dann geben wir im instrument editor ein:
[Filtertable Pos 01]
Springe zu FILT TBL [TABULATOR]
edit
[01:90 F7] (linkes nybble $9x - setze lowpass, rechtes nybble nicht verwendet,
$Fx setze resonanz auf $F,
$x7 welche stimme/oszillator gefiltert wird)
[02:00 10] (setze cutoff auf $10)
[03:FF 00] (ende des filterprogramms)

drücke jetzt [SPACE] um eine per filter gedämpfte version deines instruments zu hören.

weiterführende erklärung der byte werte:
bytes die in der linken spalte geschrieben sind definieren entweder den filtertyp oder sind teil von automatisierungs befehlen (das selbe wie bei PWM automation!). also wenn das linke byte im bereich von $01 bis $7F ist wird eine automatisierung/modulation durchgeführt, $01 bis $7F definiert die anzahl der frames wie lange moduliert wird, der dazugehörige wert in der rechten spalte der selben zeile deifiniert die geschwindigkeit/die menge die pro durchlauf hinzugefügt oder subtrahiert wird (wieder ein vorzeichenbehafteter wert).

wenn du die werte aus dem pwm automatisierungs tutorial hier ausprobierst, wirst du merken das der filter nicht rund läuft bzw. nicht konstant auf und ab geht sondern einen "harten" anfang hat im vergleich zum PWM register. der wert läuft normal von $00 bis $FF, nur ist $FF der maximal- und $00 der minimalwert, beim PWM register liegt der maximalwert in der mitte

bei $7FF. also wird es langsam wichtig zu verstehen was jetzt eigentlich ein vorzeichenbehafteter wert ist, damit man den filter auch in die andere richtung fahren kann. musikalisch kann man sagen, der filter geht "auf" desto höher der cutoff wert ist. manche behaupten desto mehr resonanz auf dem filter liegt desto mehr klingt es nach "entengequake", das macht sich vor allem am swinsidX2 bemerkbar.

wenn der bytewert in der linken spalte des filtertables über $7F (also $80 und mehr) liegt dann werden beide bytes in dieser zeile wie folgt benutzt:

filter typ (linkes byte) - da wir nur vier werte haben (kein filter/LOWpass/MIDpass/HIpass), werden nur die oberen vier bits des linken nybbles verwendet.

folgende bitkombinationen wählen den filtertyp aus:

```
     H  M  L
     I  I  O
     :  D  W
     :  :  :
% 1  0  0  0   $8 - alles aus, normaler sound ohne filter
% 1  0  0  1   $9 - LowPass - LP
% 1  0  1  0   $A - MidPass (Bandpass)
% 1  0  1  1   $B - BP & LP
% 1  1  0  0   $C - HighPass - HP
% 1  1  0  1   $D - HP & LP
% 1  1  1  0   $E - HP & BP
% 1  1  1  1   $F - HP & BP & LP
```

das rechte nybble wird nicht verwendet, also $80 oder $1, oder sogar $8F hat die selbe funktion.

das rechte byte, linkes nybble stellt die resonanz ein ($0x - $Fx), die drei untersten bits des ganz rechten nybbles gibt an welche stimme/channel/oszillator/voice gefiltert wird ($x0 - $x7):

```
    [stimme/osz.]
     3  2  1
     :  :  :
     :  :  :
     :  :  :
% 0  0  0  0   $0 - kein filter
% 0  0  0  1   $1 - stimme 1 gefiltert
% 0  0  1  0   $2 – stimme 2 gefiltert
% 0  0  1  1   $3 - stimme 1 & 2 gefiltert
% 0  1  0  0   $4 - stimme 3 gefiltert
% 0  1  0  1   $5 - stimme 1 & 3 gefiltert
% 0  1  1  0   $6 - stimme 2 & 3 gefiltert
% 0  1  1  1   $7 - alle stimmen gefiltert
```

nun können wir auch das beispiel am anfang des kapitels verstehen:

[01 90 F7] (linkes nybble $9x - setze LP, rechtes nybble ungenutzt,
$Fx setze resonanz, $x7 setze stimme/oszillator)

noch ein spezialfall: wenn das linke nybble $00 ist, so wie in step 02 unseres beispiels. [02:00 10] (setze cutoff auf $10) - $00 ist die anweisung zum direkten setzen von cutoff auf einen gewissen wert, in unserem fall $10.

ein einfacher filterverlauf:

[01:90 F7] ($90 - setze LP, $F setze resonanz, $7 setze channels)
[02:00 80] (setze cutoff auf $80 = mitte des gesamten filterbereichs)
[03:1F FF] (für $1F frames, subtrahiere $01 vom cutoff register)
[04:1F 01] (für $1F frames, addiere $01 zum cutoff register)
[05:FF 00] (ende des ablaufs)

wichtig! in den filterbeispielen wird der filter immer für alle drei stimmen ausgewählt, und zwar absichtlich - denn das instrument das du im instrument editor bearbeites wird immer am channel abgespielt in dem du zuletzt im pattern editor warst. das heisst, wenn du noten auf pattern/channel 1 eingibst und dann in den instrument editor springst und mit [SPACE] eine testnote triggerst, dann wird diese note von channel 1 / stimme 1 abgespielt, also wären hier nur filter einstellungen hörbar die sich auf kanal 1 beziehen. das heisst: wenn du einen filter bearbeitest und nichts hören kannst, kann es sein das du den klang auf dem falschen kanal bzw stimme abspielst.

falls es nicht klar aus dem tutorial hervorgeht - der sidchip hat nur einen filter - aber drei oszillatoren. wenn mehrere sounds den filter verwenden dann nimmt ein sound dem anderen den filter weg, das jeweils zuletzt gespielte instrument hat die kontrolle darüber.

sehr wichtig beim 6581 hört man beim einschalten und umschalten des filters ein lautes knacksen, man sollte also taktisch versuchen im lautesten moment des songs den filter einzuschalten, oder am beginn des songs, dann fällt das knacksen nicht so auf. wer findig ist kann auch versuchen aus dem knacksen neue sounds zu erstellen, z.b. zwischen verschiedenen filtertypen umschalten um eine art minimal sample wiedergabe zu erhalten. beim 8580 chip gibt es dieses knacksen nicht.

kapitel 4 - vibratos

kapitel 4.1 - vibratos & speedtable

in der rechten spalte des instrument editors haben wir "Vibrato Param" und "Vibrato Delay". [Vibrato Param] zeigt auf eine position im [SPEEDTBL].

[Vibrato Delay] definiert wieviele frames abgespielt werden bevor das vibrato beginnt.aber was ist ein vibrato überhaupt? ein vibrato ist eine kleine (oder grosse wenn man es so einstellt) auf und ab vibrierende änderung der tonhöhe. es gibt dem klang mehr leben, macht das der sound weniger statisch klingt. geigenspieler und gitarristen spielen oft vibratos auf ihren instrument indem sie die saite am griffbrett auf und ab bewegen. wir editieren ein instrument und machen das folgende um vibrato hinzuzufügen:

edit [Vibrato Param 01] (zeigt auf eine position im speed table)
edit [Vibrato Delay 01] (das vibrato fängt sofort an ohne verzögerung - der wert 00 schaltet das vibrato komplett ab!)

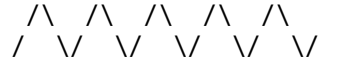
gehe zum speedtable
edit [01:0E 0F] drücke [SPACE] für ein weites (zu extremes) vibrato.
edit [01:04 05] für ein mehr unterschwelliges vibrato.

der speedtable ist etwas anders als üblichen tables. wenn er für vibrato verwendet wird (andere verwendungen im nächsten kapitel) ist das linke byte ($04 in unserem zweiten beispiel) ein indikator für wieviele frames sich die tonhöhe ändert bevor sie in die andere richtung geht. das rechte byte ($05 im zweiten beispiel) definiert die menge die zur tonhöhe addiert oder davon subtrahiert wird. desto kleiner der linke wert ist, desto schneller wird das vibrato, desto grösser der rechte wert ist, desto weiter wird die tonhöhenänderung. durch dieses konzept

mit den gezählten frames und richtungswechsel kann es auch sein das der linke wert die ursprüngliche tonhöhe ungewollt verändert. maximalwert des linken bytes ist $F7 (127 frames bis richtungswechsel), rechtes byte geht bis $FF. es gibt hier keine singed byte werte.

diagramm von [01:0E 0F]

diagramm von [01:04 05]



hinweis der speed table ist der einzige table der nicht "wie ein programm" funktioniert, er beinhaltet also keine befehle die abgearbeitet werden. jede einzelne zeile ist ein einzelner eintrag ohne nachfolge, es ist also auch nicht nötig irgendwo am ende ein $FF kommando zu setzen.

kapitel 5 - patterns & song aufbau

kapitel 5.1 - patterns editieren

da wir jetzt wissen wie man ein istrument erstellt können wir anfangen einen song zu schreiben.
wichtig zur tastaturbelegung:
+/- wechselt das instrument (idealerweise eine tastatur mit ziffernblock verwenden)

im goattracker fenster sehen wir:

```
CHN 1 PAT.00  CHN 2 PATT.01  CHN 3 PATT.02
|     |
|     derzeitiges pattern
|     das geladen is und
|     angezeigt wird
|
Sid Stimme/Channel

00 ... 00000  00 ... 00000  00 ... 00000
01 ... 00000  01 ... 00000  01 ... 00000
02 ... 00000  02 ... 00000  02 ... 00000
```

wenn wir in den pattern editor gehen und eine note eingeben (zb [N] im fasttracker layout) dann ändert sich line 00 zu:

```
00 A-2 01000
   |/| |/\ |
   | | |  \|
   | | |   pattern befehl / kommando (nichts derzeit)
   | | instrument nummer $01
   | oktave (mit / & * einstellbar)
   notenwert
```

du kannst die cursortasten verwenden um auf und ab zu scrollen und die keyboard klaviatur zu noteneingabe verwenden. desweiteren kann man [Einfg] und [Entf] verwenden um noten nach unten oder oben zu schieben. [ENTER] setzt ein keyoff, das kann verwendet werden um instrumente stumm zu schalten die das gatebit permanent ein haben. mit [shift+ENTER] wird ein keyon

gesetzt, man kann den klang also auch nach einem keyoff wieder starten. [<x] backspace] löscht die note bei cursorposition oder fügt ansonsten eine pause ein. [Bild auf/Bild ab] springt 8 schritte auf einmal im pattern.
[< und >] wechselt das pattern im channel den man gerade bearbeitet, [Einfg / Entf] am letzen schritt des patterns verändert die pattern länge. es gibt noch viele weitere shortcuts die man verwenden kann und die das komponieren einfacher gestalten, mit [F12] kann man das hilfemenü aufrufen und sich ansehen was es noch so gibt.

kapitel 5.2 - pattern befehle

die letzten drei weerte in einer pattern spalte werden für die befehle verwendet. die abkürzungen und funktionen sind:

0XY - mache nichts

1XY - portamento nach oben (tonhöhe nach oben) XY ist eine position im speedtabe. in diesem fall wird das als 16bit wert abgearbeitet, maximalgeschwindigkeit ist also $FF FF.

2XY - portamento nach unten. XY zeigt auf einen 16bit wert im speed table.

3XY - tonportamento. die tonhöhe gleitet interaktiv von einer note zur nächsten, XY zeigt wieder auf den speedtable als 16bit wert. wen XY $00 ist werden die noten zusammenhängend gespielt.

hinweis portamento wird solange fortgesetzt solange 1/2 oder 3 XY in der befehls spalte steht.

4XY - fügt vibrato zu stimme hinzu, XY zeigt auf den speedtable. wird gleicht verwendet wie instrument vibrato.

5XY - setze attack/decay register direkt auf X/Y

6XY - setze sustain/release register direkt auf X/Y

7XY - setze wellenform register direkt auf XY. wenn der wavetable der betreffenden stimme gerade die wellenform verändert hat der table vorrang.

8XY - führe wavetable ab position XY aus. $00 stoppt die ausführung.

9XY - führe pulsetable ab position XY aus. $00 stoppt die ausführung.

AXY - führe filtertable ab position XY aus. $00 stoppt die ausführung.

BXY - filter steuerung, X ist resonanz, Y ist channel auswahl
00 schaltet den filter aus und stoppt die table ausführung

CXY - setze filter cutoff auf XY - funktioniert nicht wenn filtertable bereits den filter steuert.

D0Y - stelle gesamtlautstärke ein $0-F.

EXY - shuffle tempo, XY zeigt auf den speedtable. zwischen den beiden werten im table (links & rechts) wird auf jeden taktschlag abgewechselt

FXY - setze tempo. $03-7F setzt das tempo für alle channels/patterns, $83-$FF setzt das tempo für das aktuelle pattern (subtrahiere $80 für den eigentlichen wert). $03 ist die schnellste geschwindigkeit. tempo $00 und $01 rufen das shuffle tempo auf das mit kommando EXY gesetzt wurde.

kapitel 5.3 - orderlist

die [CHN ORDERLIST] oben rechts im goattracker ist die eigentliche playlist aller patterns. drücke [F6] um dort hinzuspringen.

Normalerweise sieht es anfangs so aus:
1 00 RST00
2 01 RST00
3 02 RST00

wenn wir den cursor auf [00] bewegen und [Einfg] drücken können wir platz machen um neue patterns einzufügen, [Entf] entfernt patterns wieder. die bedienung ist gewöhnungsbedürftig: wenn man [ENTER] auf einem pattern in [CHN ORDERLIST] drückt dann wird dieses in den editor geladen. drückt man [SPACE] auf einem pattern in der [CHN ORDERLIST] dann wird es grün eingefärbt aber nicht in den editor geladen. das sind spezielle funktionen die das songwriting erleichtern. wenn wir nun folgende tasten drücken:

[F1] der song wird ganz von anfang an wiedergegeben
[F2] die grün markierten patterns werden wiedergegeben, egal was im editor geladen ist
[F3] spielt die patterns ab die gerade im pattern editor geladen und zu sehen sind.
[F4] stoppt die wiedergabe

hinweis als standard sind die steps im pattern editor dezimal (00-63) weil das musikalisch einfacher ist, pattern numbers sind aber in hex angegeben. RST00 zeigt z.b. an das die orderlist auf position 00 wiederbeginnt wenn das ende erreicht ist. so kann man den song ab gewählter position endlos wiederholen. um den song normal ausklingen zu lassen hat es sich bewährt ein leeres pattern am ende einzufügen und dieses endlos zu wiederholen.

hinweis es macht sinn sich während dem komponieren kleine notizen zu machen, welches pattern jetzt genau was macht. später, wenn man dranbleibt, hat man das alles im kopf und sieht wie in der matrix die ganzen abläufe die hinter den hex zahlen in der orderlist stehen.

kapitel 6 - gut zu wissen

kapitel 6.1 - abkürzungen, ungewöhnliches

[F4] stoppt auch den sound der im instrument editor getriggert wird

[SPACE] schaltet zwischen JAM/EDIT im instrument editor um, triggert eine testnote im intstrument editor (kann mit / und * oktaviert werden)

[F10] ladet song im pattern/orderlist editor, kann aber auch instrumente laden wenn im instrument editor [F10] gedrückt wird

[F11] speichert song im pattern/orderlist editor, speichert einzelne instrumente im instrument editor.

[ENTER] auf einem wave/pulse/filtertable Pos XY im instrumenteditor setzt dich automatisch auf diesen eintrag im betreffenden table, man muss also nicht herumsuchen wo die daten sind. wenn es ein leeres, neues instrument ist dann bringt dich [ENTER] auf den nächsten freien platz im wavetable (der erste freie platz der [$00 00] enthält).

[shift+N] im instrument editor, editiert instrumenten name

[shift+N] in tables, konvertiert bytes zu vorzeichenbehafteten werten, also kann man kapitel 0.4 jetzt abhaken, trotzdem sollte man wissen wozu man das braucht
[shift+F8] 6581/8580 emulation umschaltung

kapitel 6.2 - zusätzliche infos

wenn man einen song zum .sid kompiliert (mit [F9]) muss man aufpassen das alle daten in den tables mit einem FF befehl aufhören, sonst bekommt man einen [table execution overflow] und der song kann nicht erstellt werden.

extrem wichtig manchmal drückt man versehentlich shift lock und wundert sich warum keine eingaben mehr funktionieren, ebenso wenn man im JAM mode ist geht manchmal gar nichts. diese zwei sachen sollte man immer überprüfen wenn aufeinmal nichts mehr geht.

es ist eine gute idee den song auch mal am echten c64 probezuhören, der coole sound mit filter rafinessen den man stundenlang ausfeilt wird am echten gerät vielleicht gar nicht so gut klingen. eine alternative ist eine hardsid karte - die aber latenz- und timingprobleme hat. die billige variante ist die fakehardsid.dll simulation mit einem nokia datenkabel über rs232, da kann man den soundchip direkt, fast ohne latenz aus goattracker ansprechen und hat 100% authentisch seinen klang. im deutschen commodore forum forum64.de findet man infos über diese lösung.

---

danksagung und nachtrag

danke für unterstützung an: bordeaux, mrsid, streetuff, theryk, groepaz

persönlicher nachtrag: es war damals(tm), so ca 1991, immer mein 'traum' mit dem sidchip musik machen zu können, aber ich habe die materie nicht verstanden. erst jetzt, durch die jahre, mit dem internet und viel zeitinvestition, konnte ich viel lernen und erst durch eigenes interesse an synthesizern vieles verstehen. ich hoffe ich kann denjenigen einen start bieten die genauso gerne was am sid machen würden wie ich damals, aber es bis jetzt aufgrund komplizierter anweisungen noch nicht geschafft haben. als weiteren ansporn möchte ich noch erwähnen das ich bis heute weder noten lesen oder etwas mit musikalischen bezeichnungen anfangen kann. noten lesen ist also heutzutage absolut keine voraussetzung um musik zu machen, viele sehen es sogar als nachteil wenn die kreativität durch blattspiel und ähnliches eingegrenzt wird.

<>< unconditional/pants off, 153! - onebitman 12/2013 ><>