BLACKBIRD USER'S GUIDE

Professional music editing software
for the Commodore 64.

Contents

1

Getting started

Congratulations on your choice of **BLACKBIRD**, a modern and powerful music editor for the Commodore 64.

The **BLACKBIRD USER'S GUIDE** is designed to give you all the information you need to properly set up your equipment, quickly get acquainted with operating the **BLACKBIRD** music editor, and give you a simple, fun start at learning to make your own songs.

1.1 Introduction

BLACKBIRD belongs to a family of music editors known as *trackers*, where musical data is organised into tracks. Conceptually, **BLACKBIRD** shares many features with other tracker programs, but it also breaks new ground in a number of areas.

With the help of an external cross-platform tool, you can easily export the music that you create in **BLACKBIRD** into a SID file, an executable Commodore 64 program, or even a distributed fileset that can be streamed from disk. The exported music contains

a highly efficient *playroutine* that has a guaranteed maximum execution time of 18 rasterlines, which includes realtime decompression of the data stream. Furthermore, the playroutine has a very competitive memory footprint: In the distributed variant, the resident part of the player occupies 9–12 pages of memory, depending on the amount of different instruments and effects used.

Most of the esoteric features of **BLACKBIRD** can be traced to particular technical aspects of the highly optimised playroutine, but doing so in detail is beyond the scope of this book. For the adventurous reader, the complete source code of the playroutine is given in Appendix A.

1.2 Running the program

BLACKBIRD supports a variety of hardware configurations. This section outlines the most common setups, including instructions for loading the **BLACKBIRD** program as well as reliably saving your work. Please read the instructions for your particular setup carefully. Refer to Chapter 5 for more information about loading and saving songs.

1.2.1 Commodore 64 with external drive

To use **BLACKBIRD** on a Commodore 64 connected to an external storage unit, such as the 1541 disk drive, please proceed as follows. Turn on the power to the Commodore 64 computer and the disk drive. At the `READY` prompt, type in the following command exactly as written here:

```
LOAD "*",8
```

and press the RETURN key. When the `READY` prompt reappears,

type:

```
RUN
```

and press RETURN.

BLACKBIRD handles all saving and loading via system vectors, making full use of the acceleration routines installed by some cartridges and custom Kernal ROMs.

1.2.2 Commodore 64 with 1541 Ultimate

BLACKBIRD can also run on a Commodore 64 connected to a disk drive emulator, such as the 1541 Ultimate.

To use **BLACKBIRD** with the 1541 Ultimate, you are encouraged to activate emulation of a freezer cartridge such as Cyberpunx Retro Replay, as it will greatly accelerate loading and saving. Mount the **BLACKBIRD** disk image from the Ultimate DOS menu system. From the Cyberpunx Retro Replay boot menu, press F7 to activate filesystem acceleration. Then press F1 followed by RETURN to load the music editor.

Note in particular that you should not run the **BLACKBIRD** program file directly from the Ultimate DOS menu system, as doing so will bypass the cardridge startup code, resulting in slower filesystem access.

Please also note that drive emulators may occasionally fail to propagate saved data onto the actual storage media. You are strongly encouraged to verify that your saved songs are actually written out. On the 1541 Ultimate, the recommended procedure is to enter the Ultimate DOS menu, mount a different disk image, then navigate into the previously used disk image and verify that the saved file appears in the directory listing.

Before turning off power or removing the storage media, make it your habit to always navigate back to the root of the Ultimate DOS

browser, push the reset button and wait a few seconds.

1.2.3 Vice

It is also possible to run **BLACKBIRD** in a Commodore 64 emulator, such as *Vice*, that offers access to the hosting filesystem through a virtual device.

Be aware, however, that depending on your particular Vice setup, you might find the accuracy of the SID chip emulation to be inadequate for musical work.

At the command line prompt of the hosting operating system, navigate to the directory where you wish to keep song files, and start Vice with the **BLACKBIRD** program file as argument:

```
x64sc /path/to/blackbird.prg
```

The directory of your choice will now be available as Device 8. Since True Drive Emulation is disabled when Vice is started with a program file as argument, loading and saving will be quite fast.

Because of the way that **BLACKBIRD** makes use of the modifier keys, please ensure that the keyboard emulation in Vice is set to *positional mapping*.

1.3 Listening to the example songs

On the **BLACKBIRD** disk are a number of example songs that you may wish to study in the editor. This section takes you through the steps necessary to load an example song and listen to it. Please refer to Chapter 5 for more extensive documentation on loading and saving.

From the **BLACKBIRD** main screen, press SHIFT F1 to open the *Disk & Tools* menu. Make sure that the **BLACKBIRD** distribution

disk is in the drive, and that the correct device has been selected. The device number can be changed by pressing ↑. Then press D to obtain a directory listing.

Navigate among the listed files using the cursor keys. By convention, **BLACKBIRD** songs have filenames starting with bb. With the desired filename highlighted, press L (or RETURN) to load it.

When the file has finished loading, you may optionally press the RUN/STOP key to return to the main screen. Press F1 to start playing the song from the beginning.

While the song is playing, the current position will be highlighted in reverse video along the left side of the screen. To follow along as the song plays, press F5 to enter edit mode, and F5 again to toggle between the compact song view and the expanded track view. In edit mode, use the cursor keys to move around in the song data. You can do this while the song is playing.

Press SHIFT F3 at any time to stop playback.

2

Editing a song

To clear the current song and start from scratch, first press SHIFT F1 to enter the Disk & Tools menu, then press N for New, and finally Y to confirm. Afterwards, press RUN/STOP to get back to the main screen. You should now be looking at something like Figure 2.1.

The upper part of the display is the *combined song and track editor*. In the lower left corner is the *instrument editor* and in the lower right corner is the *effect editor*. In between the instrument and effect editors is a status area, showing the current mode (e.g. LIVE), the current octave, and four *resource counters*.

2.1 Live mode

Pressing RUN/STOP will take you to *live mode*. In this mode, you can play notes on the keyboard. The note keys are arranged in two rows, as shown in Figure 2.2. The upper row plays one octave higher than the lower row. All note keys are transposed accord-

Figure 2.1: Starting from scratch.

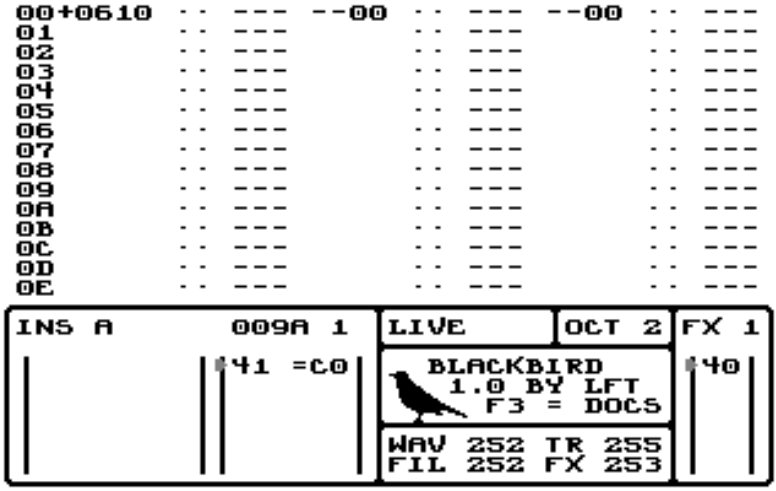
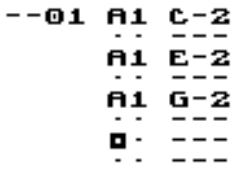


Figure 2.2: Note keys.

ing to the current octave, which can be changed using F7 and SHIFT F7. Try it now! You can also press SPACE to release the current note.

If you followed the instructions for clearing the song, you've been playing the default instrument (`INS A`, a fixed-width pulse wave) combined with the default effect (`FX 1`, a static pitch in the normal pitch range). Effects and instruments are discussed in detail in Chapters 3 and 4 respectively.

Figure 2.3: A (very short) track.



2.2 Edit mode

Press F5 to enter the combined song and track editor. Press F5 again to toggle between editing *tracks* (notes etc.) and editing the *song* (references to tracks).

The following sections will describe the meaning of the data that you'll be editing in this mode. Feel free to experiment with the default song while reading on.

Note that while you are editing the song columns, the musical data is displayed in compressed form, with one line of text representing an entire bar of music. When you switch to editing the contents of tracks, the view is expanded.

2.3 Tracks

In **BLACKBIRD**, music data is organised into tracks. A track represents one bar of music for one voice. Figure 2.3 shows how a track might be displayed in the editor. The example track is very short, 8 rows, and contains three notes. They are C, E and G of octave 2, in that order. In between the notes are blank lines, displayed as –––. When a blank line is encountered during playback, the previous note is simply held.

To the left of the notes are two columns containing an instrument name and an effect name. In the example track, all three notes

are to be played back using instrument A and effect 1. Effects and instruments are discussed in detail in Chapters 3 and 4 respectively.

On the second-to-last row of the example track, a *gate-off* command is displayed as a small square symbol in the instrument column. This command will force the gate-bit off for the remainder of the current note.

To edit a track, press F5 once or twice to position the cursor on the note column of one of the tracks. Use the cursor keys to move around inside the track, and into neighbouring tracks. Hold CTRL while pressing a cursor key to jump to the nearest track boundary in that direction.

When the cursor is in the note column, enter notes by playing them as you did in live mode (Section 2.1). Entering a note will also copy the current instrument and effect names into the appropriate columns. Press SPACE in the note column to clear the current row. Press SHIFT CLR/HOME to enter a gate-off command.

Note that in the default song, only the leftmost track is editable. We'll soon see how to change this.

The instrument and effect columns are edited directly, by simply typing the desired character. Each of them can be cleared by pressing SPACE. Clearing the instrument column produces *legato* (or *tied*) notes. Gate-off commands are entered by means of SHIFT CLR/HOME.

To pick up (select) the instrument and effect mentioned on the current track row, press ←.

The length of a track can be changed. To remove a row, press INST/DEL. To insert a new row above the current one, press SHIFT INST/DEL. To insert a new row below the current one, press CTRL INST/DEL.

The maximum length of a track is 32 rows, and this is also a typical length in practice. The minimum length of a track is one

row. The duration of a track row depends on the current *tempo*, but by default, it's about one fifth of a second. Tempo control is described in Section 2.6.

In the upper left corner of Figure 2.3 a *track reference* is shown (`--01`). The track reference is not considered part of the track. Rather, it is part of a larger structure, namely the song. The two hexadecimal digits form a *track number* that uniquely identifies this particular track.

Track 00 is special: It is always blank. Its length can be adjusted, however.

Tracks are only visible when they are part of the song. Hence, the only way to edit a track is to make it reachable from the song.

It is possible to cut, copy and paste entire tracks using CTRL X, CTRL C and CTRL V, respectively, when the cursor is located on the desired track.

To transpose all notes in a track up or down by one semitone, use CTRL + or CTRL - respectively.

Track notes are limited to the range `C-0` through `D#5` (64 semitones). Notes that are transposed beyond these bounds will wrap around to the other edge of the range, so that the action can be undone easily. Chapter 3 (effects) explains how to extend the effective pitch range.

2.4 The song

The song is constructed using tracks as building blocks. Every line of song data consists of three track references, one for each voice of the SID chip. A track reference consists of two bytes, shown as four hexadecimal digits, except that if the first byte is zero it is displayed as `--`.

As mentioned, the second byte is a track number. A track can

be referenced any number of times within the same song. This makes it easier to work with large-scale repetition, such as verse-and-chorus, and facilitates moving things around. When the final song is *exported* (see Chapter 7), **BLACKBIRD** concatenates the track data for each voice into a stream, which is then compressed. Hence, using multiple references does not save memory compared to having many identical tracks, and tracks that are nearly identical to each other nevertheless compress well together. This is a unique feature of **BLACKBIRD**.

The duration of a song line depends on the *minimum length* of the three tracks that it refers to. It is therefore possible for one or two of the tracks to be cut short during playback; this is indicated in the track editor with a colon (`:`) on the last row that is actually heard.

To edit the song, press `F5` once or twice to position the cursor on a song column. Use the cursor keys to move around, and type hexadecimal digits on the keyboard.

To remove a song line, press `INST/DEL`. To insert a line above the current one, press `SHIFT` `INST/DEL`. To insert a line below the current one, press `CTRL` `INST/DEL`.

Press `RETURN` to play the song from the current cursor position (first snapping back to the beginning of the current track). Press `F1` to play the song from the current startpoint, which is normally the beginning of the song. Press `SHIFT` `F3` to stop playback.

2.5 Transposed track references

The leftmost byte of a track reference has two different purposes. It can be used to apply a *transpose* effect to the referenced track during playback, adding or subtracting a fixed number of semitones to the notes in the track. This effect is enabled by setting the leftmost byte to a hexadecimal number in the range `40`–`c0`.

The neutral value is `80`. Thus, a value of `80` has the same effect as a value of `00` (`--`).

Notes that would be transposed outside the allowed 64-semitone range will wrap around.

Transposed track references are handy when working with e.g. bass parts, as they are often rhythmically repetitive. Due to the way **BLACKBIRD** handles exported data, memory usage is unaffected by the choice between duplicating and reusing a track, whether transposition is involved or not (see Chapter 7).

2.6 Tempo and swing

The second purpose of the leftmost byte in a track reference is to set the tempo. This is done with a hexadecimal number in the range `04`–`3f`. The second hexadecimal digit, which must be in the range `4`–`f`, controls the master tempo, expressed as the number of video frames spent per track row. A video frame is roughly 20 ms on a PAL system. The first hexadecimal digit, normally zero, is the *swing* setting, expressed as an additional number of video frames for just the even-numbered track rows.

Table 2.1 enumerates all the possible tempo settings. The default tempo setting is `06`, corresponding to a tempo of 125 bpm with no swing.

The current tempo setting is *sticky*, in the sense that it remains in effect until it is changed again. Most songs only set the tempo once, on the first song line. Note that since the tempo command is embedded in the track reference, it is only possible to set the tempo at the beginning of tracks. Should you wish to make a mid-track tempo change, you have to split the track first. Furthermore, you cannot combine a tempo command with a transpose command in the same track reference.

Table 2.1: Tempo settings, with resulting (approximate) tempo in beats per minute on a PAL system.

	0–	1–	2–	3–
–4	188 bpm	167 bpm	150 bpm	137 bpm
–5	150 bpm	137 bpm	125 bpm	116 bpm
–6	125 bpm	116 bpm	107 bpm	100 bpm
–7	107 bpm	100 bpm	94 bpm	88 bpm
–8	94 bpm	88 bpm	84 bpm	79 bpm
–9	84 bpm	79 bpm	75 bpm	72 bpm
–a	75 bpm	72 bpm	68 bpm	65 bpm
–b	68 bpm	65 bpm	63 bpm	60 bpm
–c	63 bpm	60 bpm	58 bpm	56 bpm
–d	58 bpm	56 bpm	54 bpm	52 bpm
–e	54 bpm	52 bpm	50 bpm	49 bpm
–f	50 bpm	49 bpm	47 bpm	46 bpm

2.7 The position column

Along the left edge of the combined song and track editor is the *position column*. On highlighted lines—those that contain song data—this column shows the current position within the song. On grey lines, it shows the current row number within the track.

In addition, during playback, the current playback position is visualised in this column by displaying the hexadecimal digits in reverse video.

During editing, the RETURN key will start playback from the current song position, that is, from the beginning of the track that is currently being edited. Pressing F1 in any mode will start playback from the current *startpoint*, normally song position 00. The startpoint is indicated by a plus (+) immediately to the right of the position column. It can be moved to any song position by pressing + in song editing mode. The location of the startpoint is not considered part of the song, and isn't saved.

A song can be repeating or non-repeating. When a non-repeating song reaches the end, playback stops. When a repeating song reaches the end, playback resumes from the *loop point*, which is indicated by a small arrow on the left hand side of the position column. The location of the loop point is considered part of the song, and is saved along with it. To place the loop point, press SHIFT RUN/STOP on a song column. Press it a second time to remove the loop point, making the song non-repeating again.

2.8 Empty tracks

When you start working on an empty song, all tracks are considered *empty* except—somewhat ironically—the blank track, number 00. The song may refer to any track, empty or not. As soon as you start editing the contents of a track, it becomes *non-empty*.

For a visual overview of which tracks are empty and which are non-empty, press SHIFT F1. The large square area in the middle of the screen highlights non-empty tracks. Each column corresponds to 16 consecutive track numbers.

Empty tracks have a special property: Their length is flexible, and adapts according to the lengths of neighbouring tracks in the song. This makes it easier to work with non-standard time signatures. As soon as an empty track is modified, it becomes non-empty and its length becomes fixed.

As a special case, if a song line only refers to empty tracks, they will mimic the length of track 00.

To revert a track to empty status, cut the track by pressing CTRL X in track editing mode. Chapter 6 also describes a technique for quickly emptying all unreferenced tracks.

3

Effects

An *effect* in **BLACKBIRD** is simply a table of pitch offsets in the range `00`–`7f`, where each row has a duration of one video frame. The pitch offsets are expressed in *microtones* ($\frac{1}{4}$ of a semitone), and the default offset is the hexadecimal number 40.

On every video frame, the pitch that is actually emitted by the SID chip is the sum of the pitch given by the most recent track note and the current value from the effect table. By using values that are lower or higher than the default (40), it is possible to escape from the range of 64 semitones offered by the track editor.

For instance, in a blank song, the default effects `0`, `1` and `2` have been prepared with the hexadecimal values 10, 40 and 70, respectively. This corresponds to three different pitch ranges: Subbass, normal and treble. Notice that the difference between these numbers is 30 (that is 48 in decimal notation), which is precisely four times the number of semitones in an octave. Thinking in microtones can be tricky at first, but one quickly gets the hang of it.

Note that if the sum of a track note and an effect value exceeds

the total pitch range allowed by the SID chip, a random pitch will be heard instead. This won't happen as long as the effect value is near 40, but it can occur when using effect values closer to 70.

As a special case, a value of `00` in the effect table always selects the highest pitch possible on the SID chip (i.e. a register value of `ffff`), regardless of the current track note. This is typically used to add a brief noise burst at the beginning of notes.

By convention, effect ↑ is used for gliding up to a note, and effect ∗ is used for vibrato. You have to put numbers into these effects yourself, however, which puts you in charge of what these expressions sound like in a given song. Please refer to the songs included on the **BLACKBIRD** distribution disk for several examples.

When a track is played back, every new note causes a retrigger of the last effect that was in use. In other words, it is not possible to change the pitch (as given by the track note) without restarting an effect. On the other hand, it is perfectly possible to restart an effect (or start a new one) without changing the pitch or retriggering the instrument. This is often done with vibrato effects, in particular.

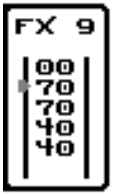
3.1 Editing effects

The effect editor is located in the bottom right corner of the main screen. In Figure 3.1, it is displaying an effect that starts at maximum pitch (to be combined with a noise waveform in the instrument) and then goes into a loop of two pitches, an octave apart.

A **BLACKBIRD** song can make use of up to 48 different effects. They have the following 48 single-character names, each corresponding to a key on the keyboard:

0 1 2 3 4 5 6 7 8 9 A B C D E F G H I J K L M N
O P Q R S T U V W X Y Z , . / : ; = @ * ↑ + – £

Figure 3.1: A half-speed octave arpeggio to be used with an initial noise burst.



To select an effect, hold the **left** SHIFT key while typing one of the effect names from the list above. To pick up (select) an effect from the current track row, press SHIFT ←.

To enter the effect editor, press SHIFT F5. If this is done while a blank effect is selected, a first row is automatically added.

To remove a row in the effect editor, press INST/DEL. To insert a row above the current one, press SHIFT INST/DEL. To insert a row below the current one, press CTRL INST/DEL.

To set the loop point for the effect, press SHIFT RUN/STOP.

It is possible to cut, copy and paste entire effects using CTRL X, CTRL C and CTRL V, respectively.

The entire effect can be transposed up or down in microtone steps using CTRL + and CTRL -, respectively.

3.2 Automatic arpeggio generation

Arpeggios are ordinary effects. However, it would be rather cumbersome to edit such effects manually, as doing so would involve performing hexadecimal arithmetic while counting in microtones. But have no fear: **BLACKBIRD** has a feature that makes it very straightforward to work with arpeggios.

To create an arpeggio, simply hold the **right** SHIFT key and play a sequence of notes, starting with the highest. As soon as you release the **right** SHIFT key, **BLACKBIRD** will find or create an effect describing the desired arpeggio. Afterwards, you'd typically press the highest note key once more, to actually insert an arpeggiated note into the current track.

When **BLACKBIRD** needs to allocate a name for a new arpeggio, it first tries to pick an unused letter from the alphabet. If that fails, it tries the numbers, and finally the remaining symbols.

Note that arpeggios have a limited span, i.e. there's a maximum difference between the highest and the lowest note. The first note you type in will always be represented by the value 70 at the top of the effect, and the remaining values will be filled in according to their relative pitch compared to that note. That is why you should generally begin with the highest note.

Please observe that arpeggios entered in this way will play back at a rate of one note per video frame. There is currently no convenient way of working with slower arpeggios, but they can of course be constructed manually by duplicating rows.

When entering notes into an arpeggio, it is possible to switch to a lower octave in the normal way, by pressing SHIFT F7. However, it is not possible to go back to a higher octave, since SHIFT is being held.

3.3 Resource management

In an exported **BLACKBIRD** song, all effects need to share a single 255-byte table. An effect requires one byte of table space per row, plus one for the effect itself. Hence, it may be necessary to keep an eye on the number of free bytes in this table. One of the resource counters on the main screen always displays the number of bytes left in the effect table.

4

Instruments

The *instrument editor* (Figure 4.1) is located in the bottom left corner of the main screen.

An *instrument* in **BLACKBIRD** comprises three parts: A volume envelope (displayed in the top right corner of the instrument editor), a wavetable (bottom right) and optionally a filter table (bottom left). We will discuss each part in turn.

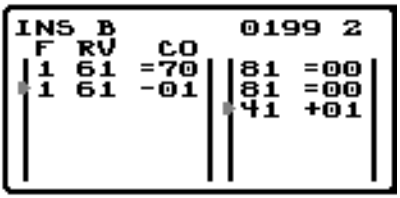
A song can make use of up to 48 different instruments. They have the following 48 single-character names, each corresponding to a key on the keyboard:

0 1 2 3 4 5 6 7 8 9 A B C D E F G H I J K L M N
O P Q R S T U V W X Y Z , . / : ; = @ * ↑ + – £

To select an instrument, hold the C= key while typing one of the effect names from the list above. To pick up (select) an instrument from the current track row, press C= ←.

To enter the instrument editor, press C= F5. If this is done while a blank instrument is selected, a first row is automatically added to the wavetable.

Figure 4.1: The instrument editor. The row saying "F RV CO" is only visible when the filtertable is enabled for the current instrument (see Section 4.3). It serves as a quick reminder of what the various columns in the table do.



To enable the filtertable for the current instrument, press CTRL F. To disable the filtertable again, remove all rows from it.

To remove a row from the wavetable or filtertable, press INST/DEL. To insert a row above the current one, press SHIFT INST/DEL. To insert a row below the current one, press CTRL INST/DEL.

Press SHIFT RUN/STOP to set the loop point in the wavetable or filtertable.

It is possible to cut, copy and paste entire instruments by pressing CTRL X, CTRL C and CTRL V, respectively.

4.1 Envelope

In the top right corner of the instrument editor are five hexadecimal digits that control the volume envelope. This area can be reached using the cursor keys.

The first four digits are the familiar Attack, Decay, Sustain and Release parameters.

The fifth digit controls the *hard-restart* feature for the current instrument. There are three modes:

0: No hard-restart

Notes are triggered normally, by setting the gate bit. This mode will expose you to the infamous envelope bugs in the SID chip:

- If the attack of the new note is less than the release of the previous note, **or**
- if the attack of the new note is less than the decay of the new note, **or**
- if the release of the new note is less than the release of the previous note

then the attack phase may be randomly delayed by up to 33 ms (1.67 video frames).

Note that if the gate bit was already set, no new attack phase is triggered. Hence, this mode is useful if you want to switch to a different wavetable or filtertable halfway through a note.

The registers are written in the following order in this mode: AD, SR, WF.

1: Normal hard-restart

Two frames before every note, the gate bit and the SR register are cleared. This guarantees that the new note starts immediately. Because the hard-restart is performed ahead of time, the note start will sync up with notes that are played without hard-restart, as well as legato notes (where the instrument column has been cleared with SPACE).

2: Double hard-restart

Like normal hard-restart, but also deliberately triggers an envelope bug at the note start, delaying the attack by 33 ms. This

means that the first row of the wavetable, filtertable and effect is ignored, and the second row only plays during $\frac{1}{3}$ of a frame, while the volume is rising. This can be used to create very snappy noise bursts at the beginning of notes. However, these notes no longer sync up with legato notes or notes that are played with normal or no hard-restart, so use this mode carefully.

The example instrument in Figure 4.1 uses double hard-restart. Notice the two rows of `81` in the wavetable. The first is never heard, and the second creates a brief noise burst.

Double hard-restart only works for attack rates 0 and 1, otherwise **BLACKBIRD** falls back on normal hard-restart.

4.2 The wavetable

The *wavetable* defines how the SID control register is to be updated over time. It is also in charge of the pulse-width register.

Each row in the wavetable corresponds to one video frame. The two leftmost columns determine the waveform. Typical waveforms are listed in Table 4.1, but you may of course use hardsync and ring modulation as well. For more details, please refer to the SID chip documentation in the *Commodore 64 Programmer's Reference Guide*.

It is possible to turn off the gate directly from the wavetable, by leaving the least significant bit unset. The effect of a gate-off command in the track data is to force the gate bit to be zero until the next time an instrument is triggered.

The remaining three columns manipulate the *pulse-width*. These columns have no effect on rows where bit 6 of the control register is unset. As a reminder of this fact, they are displayed in a different colour on such lines.

The first of the three pulse-width columns is a command, and the

Table 4.1: Typical control register values. Note that the combined waveforms sound differently on the old and new versions of the SID chip.

11	Triangle
21	Sawtooth
41	Pulse
81	Noise
31	Combined waveform 31
51	Combined waveform 51
61	Combined waveform 61
01	Silence

two remaining columns are its parameter. There are two commands:

= sets the pulse-width.

+ adds an offset to the current pulse-width value, wrapping around from ff back to 00. The maximum offset that can be added is 7f.

The current pulse-width is maintained as a byte value, but this value doesn't get written directly to the SID chip. Instead, it is reinterpreted as follows:

A pulse-width value of 80 represents a square wave (50% duty cycle). The extreme values 00 and ff represent a very narrow pulse wave (97% duty cycle). Both ends of the pulse-width range represent the same register value (both are 97% and neither is 3%), which means that there is no audible click when wrapping.

To remove a row from the wavetable, press INST/DEL. To insert a row above the current one, press SHIFT INST/DEL. To insert a row below the current one, press CTRL INST/DEL.

To set the loop point, press SHIFT RUN/STOP.

4.3 Using the filter

The SID filter is controlled globally by a filtertable. An instrument may *install* a new filtertable when triggered. The installed table remains in effect even after the current instrument has stopped playing. It gets replaced the next time an instrument with a filtertable (possibly the same instrument) is triggered.

You can manually reset the filter unit by pressing CTRL R. This uninstalls the currently active filtertable.

It doesn't matter from which voice a filtertable is installed. Its effect is global, and the filter routing is controlled from within the table. This makes it possible to manipulate the filter from one voice, and have it apply to another voice (or several voices, although this doesn't work well on the old version of the SID chip).

The three leftmost columns in the filtertable control the filter type (Table 4.2), resonance (`0`–`f`) and voice routing (Table 4.3), respectively.

The three rightmost columns control the *cut-off frequency*. The first of these is a command, and the remaining two are its parameter. There are three commands:

= sets the cut-off to a given value. The valid range is `00`-`ff`.

\+ adds an offset to the current cut-off value unless doing so would cause a wraparound.

\- subtracts an offset from the current cut-off value unless doing so would cause a wraparound.

Note a subtlety in the wording above: Suppose the current cut-off value is `fe` and an attempt is made to increment it by two (`+02`). After this operation, the cut-off value remains at `fe`; in particular, it doesn't go to the limit (`ff`).

The example filtertable in Figure 4.1 sets up a lowpass filter with resonance `6` affecting voice 1. The cut-off value is initially set to

Table 4.2: Filter types.

1	Lowpass
2	Bandpass
3	Wide lowpass
4	Highpass
5	Notch
6	Wide highpass
7	Allpass

Table 4.3: Filter routing.

0	No filter
1	Filter voice 1
2	Filter voice 2
4	Filter voice 3
3	Filter voices 1 and 2
5	Filter voices 1 and 3
6	Filter voices 2 and 3
7	Filter voices 1, 2 and 3

70, and is then decremented repeatedly. Hence, the decrementing will go on for 70 frames (in hexadecimal) and then stop.

One row in the filtertable always corresponds to one video frame. Suppose you want to decrement the filter for a long time, then increment it for a long time, then decrement again, etc. In **BLACKBIRD**, unlike many other trackers, this has to be handled from the track data. Make two copies of the current instrument, but let one of them sweep the filter upwards, and the other downwards. Change the hard-restart setting to 0. Now you can call upon these instruments by directly editing the instrument column in the track data.

4.4 Resource management

In an exported **BLACKBIRD** song, all instruments need to share a single 255-byte wavetable and 252-byte filtertable. An instrument requires one or two bytes of wavetable space per row, plus one for the instrument itself. Each row in the filtertable requires three bytes, plus one for the instrument itself, if it makes use of the filter at all.

Hence, it may be necessary to keep an eye on the number of free bytes in these tables. Two resource counters on the main screen always display the number of bytes left in these tables.

5

Disk operations

Naturally, **BLACKBIRD** can save and load songs, and it knows how to make use of most filesystem acceleration routines provided by cartridges and custom Kernal ROMs.

Press SHIFT F1 to bring up the Disk & Tools menu, shown in Figure 5.1.

From here, press N to clear the current song and start from scratch.

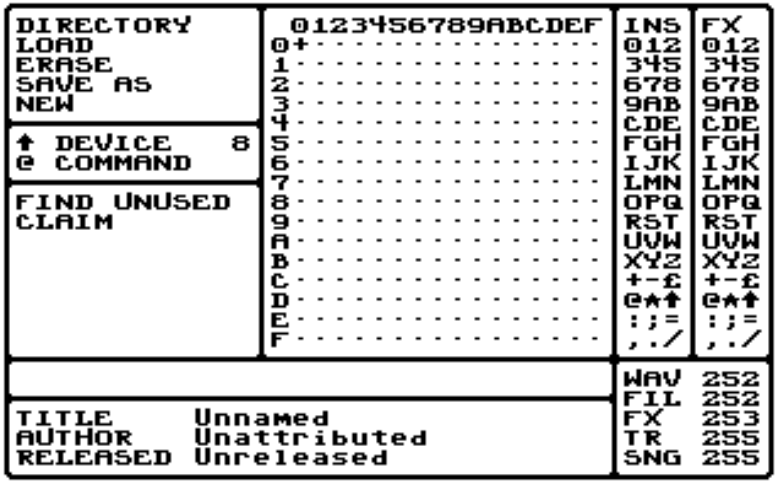
Press ↑ to change the current IEC device.

Press @ to send a command to the IEC device. Enter a blank command to read out the device status.

Press S to save the current song to disk. You will be asked for a filename. To abort the save operation at this point, press RUN/STOP in the line editor.

Press D to call up the filesystem directory of the currently selected device. The display will now look something like Figure 5.2. At the top, **BLACKBIRD** displays the name of the disk and the number of free blocks. You may refresh the directory display at any time by pressing D again.

Figure 5.1: The Disk & Tools menu.

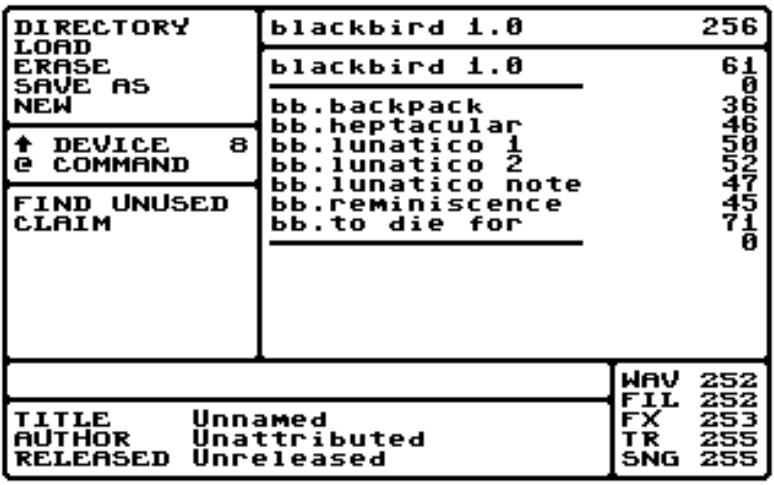


Unless the current directory was completely empty, you will find that the topmost filename is displayed in a highlighted colour. Use the cursor keys to navigate among the filenames.

Press L or RETURN to load the highlighted song.

Press E if you wish to delete the highlighted file. This will merely take you to the @-command prompt, preloaded with "S0:" followed by the selected filename. You may abort the delete operation at this point by pressing RUN/STOP.

Figure 5.2: Viewing a directory listing. There are 256 free blocks on this disk.



6 Advanced tools

Press SHIFT F1 to bring up the Disk & Tools menu, shown in Figure 5.1.

6.1 Editing song metadata

Towards the bottom of the screen are three fields capable of holding up to 32 characters of arbitrary text each. They are called *Title*, *Author* and *Released*, and they are saved along with the song. When the song is exported to a SID file, these fields are copied into the corresponding header fields.

The fields are edited by pressing T, A or R respectively.

6.2 Reclaiming unused resources

The main part of the Disk & Tools screen is occupied by visualisations of track, instrument and effect usage. The highlighted

items are non-empty. However, it is possible that a non-empty resource isn't used anywhere in the song, so that its memory can be reclaimed.

There is no need to reclaim unused resources prior to exporting (see Chapter 7), as the external tool will do this automatically. However, if you are running up against some of the built-in limits of **BLACKBIRD**, being able to easily clean out the old stuff can be of great importance.

Press F to locate unreferenced tracks.

Press SHIFT F to locate unreferenced effects.

Press C= F to locate unreferenced instruments.

Unused resources will be highlighted in reverse video. Press C to delete them and reclaim the memory that they were using.

Note that removing tracks might remove the last references to some of the other resources. Hence it is often a good idea to find and reclaim unused tracks first, and then find and reclaim unused instruments and effects.

7

Exporting with Birdcruncher

Once a song is finished, the need arises to export the music into an executable format, such as a SID file or a runnable program.

BLACKBIRD comes with a highly optimised playroutine with a guaranteed maximum execution time of 18 rasterlines. The memory footprint of the player is also quite small, although it does require 16 consecutive bytes of zero-page storage.

The **BLACKBIRD** playroutine is innovative in that it does away with the traditional track-based storage format. Instead, the note data for each voice is stored as a long run-length encoded sequence, and the three sequences are interleaved into a combined stream. This stream is then crunched using a variant of Lempel-Ziv compression featuring a copy-with-transpose primitive.

Songs cannot be exported by the **BLACKBIRD** software directly, since a memory-heavy crunching operation is necessary. Instead, songs are exported using a cross-platform tool called *Birdcruncher*, that is included in the **BLACKBIRD** package. The program is dis-

tributed as C source code for UNIX-like operating systems, and a pre-built binary is included for `win32`-compatible platforms such as *Wine*.

Run Birdcruncher without arguments to see a list of options.

Birdcruncher supports three different output formats, referred to by the tool as `sid`, `prg` and `dist`. Each format will now be described in turn.

Regardless of what output format you choose, you are encouraged to run Birdcruncher with the `-v` option, enabling verbose mode.

7.1 SID files

Birdcruncher can create SID files. The metadata fields (Title, Author, Released) are included in the file. The SID header contains a field for specifying the recommended SID model. By default, Birdcruncher sets this to be the new chip (8580). Use the `-0` option to instruct Birdcruncher to request the old chip (6581) instead.

The address of the playroutine can be given with the `-a` option, and the address of the 16-byte zero-page area needed by the player can be given with `-z`.

7.1.1 Syncpoints

Syncpoints may optionally be specified on the commandline or in a separate file. They allow external software to synchronise with the music.

Syncpoints are positions within the song (written as `SS:TT` where `SS` is the song position and `TT` is the track position). They can be given directly using the `-s` option, or read from a file named by the `-@` option. Syncpoints in a file have to be prefixed by the `@` character.

Synchronisation is carried out through a byte in the zero-page, specifically thirteen bytes into the declared zero-page area. Normally, this is location `ed`.

The synchronisation byte is cleared by the init routine. It needs to be incremented to 1 whenever the external software expects a syncpoint. When the syncpoint is reached, the playroutine resets the byte back to 0.

Should the playroutine reach a syncpoint while the synchronisation byte is 0, playback will pause until the byte contains 1. This is useful in trackmos to suspend playback in case of I/O trouble.

7.2 Runnable PRG files

Birdcruncher can also produce executable Commodore 64 programs. This output format is not as versatile as the SID format, but it is provided anyway as a convenience. The metadata (Title, Author, Released) is displayed on the screen while the song plays.

Three options allow you to customise the look of the player: `-f`, `-b` and `-r` set the colour of the foreground, background and raster-time visualiser, respectively.

7.3 Distributed output

The most powerful aspect of Birdcruncher is the ability to split the output into multiple files. The `dist` output format supports streamed playback, and is useful for trackmos where the music player may be bound by very tight memory constraints.

The distributed version of the player consists of a resident part, the init routine and the datastream. The resident part contains the playback code as well as the instrument and effect data. The

init routine contains code that can be overwritten after it has been called.

For convenience, the playroutine is located three bytes into the resident part. The first three bytes are a `jmp` instruction pointing to the init routine.

The datastream can be split into multiple files. This is done using syncpoints. Ordinary syncpoints were described in Section 7.1.1, under the `sid` output format. The distributed output format supports a special kind of syncpoint that indicates the beginning of a new *chunk* of the stream. Syncpoints of this kind are written `SS:TT:AAAA:Filename` where `SS` is the song position, `TT` is the track position and `AAAA` is the load address of the new chunk. The first chunk (and thus the first syncpoint) must be at position `00:00`.

As with the `sid` output format, the external software uses a zero-page byte (normally at `ed`) to synchronise with the playroutine.

An example showing how to use the distributed output format is included in the **BLACKBIRD** package.

8

Keyboard commands

This chapter summarises all keyboard commands used by **BLACKBIRD** (on the main screen).

The middle column indicates in what mode(s) the command is applicable. It should be interpreted as follows:

- `L` Live mode
- `S` Song editor
- `T` Track editor
- `I` Instrument editor
- `E` Effect editor

Key	Mode(s)	Action
cursor keys	`-STIE`	Move cursor
`CTRL` *cursor keys*	`-ST-`	Move trackwise
`CLR/HOME`	`-STIE`	Move to top
`C=` *name*	`LSTIE`	Select instrument *name*
Left `SHIFT` *name*	`LSTIE`	Select effect *name*
Right `SHIFT` *notes*	`LSTIE`	Auto arp (see Section 3.2)
`←`	`-T-`	Pick up instrument and effect

Key	Mode(s)	Action
C= ←	-T-	Pick up instrument
SHIFT ←	-T-	Pick up effect
F7	LSTIE	Octave up
SHIFT F7	LSTIE	Octave down
note key	L--	Play live note
SPACE	L--	Release live note
RETURN	-ST-	Play from here
+	-S--	Set startpoint
F1	LSTIE	Play from startpoint
SHIFT F1	LSTIE	Disk & Tools menu
F3	LSTIE	Online documentation
SHIFT F3	LSTIE	Stop playback
RUN/STOP	LSTIE	Stop and go to live mode
CTRL R	LSTIE	Reset SID and filter
F5	LSTIE	Edit track/song
C= F5	LSTIE	Edit instrument
SHIFT F5	LSTIE	Edit effect
literal key	-STIE	Enter data
SPACE	-ST-	Clear current field
SHIFT CLR/HOME	-T-	Enter gate-off command
SHIFT RUN/STOP	-S-IE	Set loop point
INST/DEL	-STIE	Delete current row
SHIFT INST/DEL	-STIE	Insert row before
CTRL INST/DEL	-STIE	Insert row after
CTRL C	-TIE	Copy
CTRL X	-TIE	Cut (and mark empty)
CTRL V	-TIE	Paste
CTRL +	-T-E	Transpose up
CTRL -	-T-E	Transpose down
CTRL F	LSTIE	Activate filtertable

A

The playroutine

For your reference, this appendix contains the complete source code for version 1.2 of the **BLACKBIRD** playroutine.

```
 1                  .ext    unpackbufs
 2                  .ext    zp_base
 3
 4                  .ext    INS_RESTART
 5                  .ext    INS_RESTART2
 6
 7                  .ext    fxtable
 8                  .ext    wavetable
 9                  .ext    filttable
10
11                  .ext    fx_start
12                  .ext    ins_ad
13                  .ext    ins_sr
14                  .ext    ins_wave
15                  .ext    ins_filt
16
17                  .ext    streamstart
18
19  zp_bufs         = zp_base+$0    ; words at 0-1, 7-8, e-f
20  zp_inptr        = zp_base+$2    ; word
21  zp_trwpos       = zp_base+$4
22  zp_pendoob      = zp_base+$5
23  zp_master       = zp_base+$6
24  zp_filtpos      = zp_base+$9
25  zp_tempo        = zp_base+$a
26  zp_extsync      = zp_base+$d
27
```

```
28  ; ================================================================================
29  ; Encoding (in order of appearence)
30  ;       f9-ff   Out-of-band data                only in voice 3
31  ;       c9-f8   Arpeggio
32  ;       80      Gate off                        \_ at most one
33  ;       81      Legato                          |
34  ;       83-b2   Instrument                      /
35  ;       00-7f   Note (lsb is delay-bit)         \_ at most one
36  ;       b8-c7   Delay (low 4 bits)              /
37  ; ================================================================================
38
39  ; ================================================================================
40  ; Entry points at * and *+3
41  ; (12 for jsr/rts)
42  ; Prepare1, 12 + 17 + 336 + 165 + 563   = 1093
43  ; Prepare2, 12 + 17 + 336 + 197 + 563   = 1125
44  ; Prepare3, 12 + 17 + 336 + 206 + 563   = 1134
45  ; Execute, 12 + 9 + 541 + 563           = 1125
46  ; --> 18 rasterlines max
47  ; ================================================================================
48
49  #if REPEAT
50                  .seg    seg_rplay
51  #else
52                  .seg    seg_play
53  #endif
54
55  playorg
56                  jmp     initroutine
57  playroutine
58                  .(
59                  lax     'zp_master
60                  beq     do_execute
61
62                  sbx     #7
63  +stx_unpackvoice
64                  stx     'zp_master
65                  cpx     #3*7
66                  bcs     nounpack
67
68                  jmp     unpackvoice
69  nounpack
70                  jmp     everyframe
71  do_execute
72                  jmp     execute
73                  .)
74
75  ; ================================================================================
76  ; Run timer, fetch oob and fx commands
77  ; 63 + 2 * 47 - 1 + 9 = 165
78  ; ================================================================================
79
80  prepare1
81                  .(
82  vloop
83                  inc     v_trtimer,x
84                  bmi     vskip
85
86                  lda     (zp_bufs,x)
87                  cmp     #$f9
88                  bcc     no_oob
89
90                  sta     'zp_pendoob
91                  inc     'zp_bufs,x
92                  lda     (zp_bufs,x)
93                  clc
94  no_oob
95                  sbc     #$c8-1
96                  bcc     no_fx
```

```
 97
 98                  inc     ‘zp_bufs,x
 99                  sta     v_currfx,x
100                  sta     v_pendfx,x
101  no_fx
102  vskip
103                  txa
104                  sbx     #7
105                  bpl     vloop
106
107                  lda     #<prepare2
108                  sta     preparejmp+1
109                  jmp     everyframe
110                  .)
111
112  ; ============================================================================
113  ; Check timer, fetch ins command, peek at note command, do hard restart
114  ; 3 * 63 - 1 + 9 = 197
115  ; ============================================================================
116
117  prepare2
118                  .(
119  vloop
120                  lda     v_trtimer,x
121                  bmi     vskip
122
123                  lda     (zp_bufs,x)
124                  bpl     got_note
125
126                  cmp     #$b8
127                  bcs     vskip
128
129                  inc     ‘zp_bufs,x
130                  sbc     #$82-1
131
132                  bmi     got_special
133
134                  sta     v_currins,x
135  noteback
136                  sta     v_pendins,x
137
138                  cmp     #INS_RESTART+1
139                  bcc     norestart
140
141                  lda     #0
142                  sta     $d406,x
143                  lda     #$fe
144                  sta     v_wavemask,x
145  norestart
146  vskip
147                  txa
148                  sbx     #7
149                  bpl     vloop
150
151                  lda     #<prepare3
152                  sta     preparejmp+1
153                  jmp     everyframe
154  got_note
155                  lda     v_currins,x
156                  bpl     noteback        ; always
157  got_special
158                  sta     v_pendins,x
159                  bmi     vskip           ; always
160                  .)
161
162  ; ============================================================================
163  ; Check timer, fetch note or delay-command
164  ; 3 * 69 - 1 = 206
165  ; ============================================================================
```

```
166
167   prepare3
168                   .(
169   vloop
170                   lda     v_trtimer,x
171                   bmi     vskip
172
173                   lda     (zp_bufs,x)
174                   bmi     got_delay
175
176                   lsr
177                   sta     v_pendnote,x
178
179                   lda     v_pendins,x
180                   bne     alreadyins      ; gate-off or legato
181
182                   lda     v_currins,x
183                   sta     v_pendins,x
184   alreadyins
185                   lda     v_currfx,x
186                   sta     v_pendfx,x
187
188                   lda     #$ff
189                   rol
190   got_delay
191                   ora     #$f0
192                   sta     v_trtimer,x
193                   inc     'zp_bufs,x
194   vskip
195                   txa
196                   sbx     #7
197                   bpl     vloop
198
199                   ;jmp    everyframe
200                   .)
201
202   ; ============================================================================
203   ; Code that runs on each frame. Reads the fx-, wave- and filter tables.
204   ; 2 + (31 + 46 + 89) * 3 - 2 + 65 = 563 cycles
205   ; ============================================================================
206
207   everyframe
208                   .(
209                   ldx     #14
210   vloop
211                   ldy     v_fxpos,x
212                   lda     fxtable+1,y
213                   bmi     dofxjump
214
215                   lda     #0
216   dofxjump
217                   sec
218                   adc     v_fxpos,x
219                   sta     v_fxpos,x
220                   lda     fxtable,y
221                   beq     fixedfreq
222           ; 31
223                   .(
224                   clc
225                   adc     v_basepitch,x
226                   ror
227                   bcc     fractional_x0
228   fractional_x1
229                   lsr
230                   tay
231                   bcc     fractional_01
232   fractional_11
233                   lda     freq_lsb,y
234                   ; no clc, adds small consistent error
```

```
235                     adc     freq_lsb+19+1,y
236                     sta     $d400,x
237                     lda     freq_msb,y
238                     adc     freq_msb+19+1,y
239                     jmp     freqdone1
240     +fixedfreq
241                     lda     #$ff
242                     sta     $d400,x
243                     bmi     freqdone1       ; always
244     fractional_01
245                     lda     freq_lsb+19,y
246                     ;clc
247                     adc     freq_lsb+1,y
248                     sta     $d400,x
249                     lda     freq_msb+19,y
250                     adc     freq_msb+1,y
251                     jmp     freqdone1
252     fractional_x0
253                     lsr
254                     tay
255                     bcs     fractional_10
256     fractional_00
257                     lda     freq_lsb+24,y
258                     sta     $d400,x
259                     lda     freq_msb+24,y
260                     jmp     freqdone1
261     fractional_10
262                     lda     freq_lsb+12,y
263                     ; no clc, adds small consistent error
264                     adc     freq_lsb+12+1,y
265                     sta     $d400,x
266                     lda     freq_msb+12,y
267                     adc     freq_msb+12+1,y
268     freqdone1
269                     sta     $d401,x
270     freqdone
271                     .)
272             ; 46
273                     ldy     v_wavepos,x
274                     lda     wavetable,y
275                     cmp     #$c0
276                     bcc     nojump
277
278                     ;sec
279                     adc     v_wavepos,x
280                     tay
281                     lda     wavetable,y
282     nojump
283                     and     v_wavemask,x
284                     sta     $d404,x
285                     asl
286                     bpl     nopulse
287
288                     tya
289                     ;clc
290                     adc     #2
291                     sta     v_wavepos,x
292
293                     lda     wavetable+1,y
294                     bmi     pwset
295
296                     ;clc
297                     adc     v_pwidth,x
298                     .byt    $80     ; nop imm, eats the asl
299     pwset
300                     asl
301
302                     sta     v_pwidth,x
303                     tay
```

```
304                     lda     pwprepare,y
305                     sta     $d402,x
306                     sta     $d403,x
307     postpulse
308                     txa
309                     sbx     #7
310                     bmi     vdone
311
312                     jmp     vloop
313     nopulse
314                     iny
315                     tya
316                     sta     v_wavepos,x
317                     jmp     postpulse
318             ; 89
319     vdone
320             ; * 3 - 2
321                     .)
322
323                     .(
324                     ldy     'zp_filtpos
325                     lda     filttable+3,y
326                     bmi     filtjump
327
328                     lda     #2
329     filtjump
330                     sec
331                     adc     'zp_filtpos
332                     sta     'zp_filtpos
333
334                     lda     filttable,y
335                     sta     $d418
336                     lda     filttable+1,y
337                     sta     $d417
338                     lda     filttable+2,y
339                     asl
340                     bcs     coset
341
342                     cmp     #$80
343                     ror
344                     ;clc
345     +m_cutoff       = * + 1
346                     adc     #$80
347                     bvs     filtdone
348     coset
349                     sta     m_cutoff
350                     eor     #$80
351                     sta     $d416
352     filtdone
353                     .)
354             ; 65
355                     rts
356
357     ; ===========================================================================
358     ; Execute pending events.
359     ; 136 + 385 + 20 = 541
360     ; ===========================================================================
361
362     sync_error
363                     ; main thread not waiting for sync. i/o trouble? hold playback
364                     jmp     everyframe
365     execute
366                     .(
367                     lda     'zp_pendoob
368                     lsr
369                     bcc     no_sync
370
371                     lsr     'zp_extsync
372                     bcc     sync_error
```

```
373   no_sync
374                   lsr
375                   bcc     no_tempo
376
377                   tax
378                   lda     'zp_inptr
379                   ;sec
380                   sbc     #2
381                   sta     'zp_inptr
382                   bcs     noc1
383
384                   dec     'zp_inptr+1
385   noc1
386                   ldy     #2
387                   lda     (zp_inptr),y
388                   sta     'zp_tempo
389                   dey
390                   lda     (zp_inptr),y
391                   sta     m_groove
392                   txa
393   no_tempo
394                   lsr
395                   bcc     no_eos
396
397                   lda     'zp_inptr
398                   ;sec
399                   sbc     #2
400                   sta     'zp_inptr
401                   bcs     noc2
402
403                   dec     'zp_inptr+1
404   noc2
405                   ldy     #2
406                   lda     (zp_inptr),y
407                   tax
408                   dey
409                   lda     (zp_inptr),y
410                   sta     'zp_inptr
411                   stx     'zp_inptr+1
412   #if REPEAT
413                   lda     'zp_pendoob
414                   and     #1
415                   bne     norepeat
416
417                   ldx     'zp_bufs+14
418                   stx     v_trwpos+14
419                   lax     'zp_bufs+7
420                   sbx     #256-7
421                   stx     v_trwpos+7
422                   lax     'zp_bufs+0
423                   sbx     #256-7
424                   stx     v_trwpos+0
425   norepeat
426   #endif
427   no_eos
428                   lda     #0
429                   sta     'zp_pendoob
430                   .)
431           ;108 - 4 (at most one page crossing) + 32 (repeat)
432
433                   .(
434                   ldx     #14
435   vloop
436                   lda     v_pendnote,x
437                   asl
438                   asl
439                   sta     v_basepitch,x
440
441                   ldy     v_pendfx,x
```

```
442                 beq     no_fx
443
444                 lda     fx_start-1,y
445                 sta     v_fxpos,x
446 no_fx
447                 ldy     v_pendins,x
448                 beq     ins_done
449
450                 bpl     no_special
451
452                 tya
453                 cmp     #$fe            ; fe = gate off, ff = legato
454                 bne     ins_done
455
456                 sta     v_wavemask,x
457                 beq     ins_done        ; always
458 no_special
459         ; 37
460                 cpy     #INS_RESTART2+1
461                 bcc     restart01
462
463                 lda     #$0f
464                 sta     $d406,x
465 restart01
466                 lda     #$ff            ; counter = 0..9
467                 sta     v_wavemask,x    ; counter = 2..11
468
469                 lda     ins_filt-1,y    ; counter = 7..16
470                 beq     nograbfilt      ; counter = 11..20
471
472                 sta     'zp_filtpos     ; counter = 13..22
473 nograbfilt
474                 lda     ins_wave-1,y    ; counter = 14..25
475                 sta     v_wavepos,x     ; counter = 18..29
476
477                 cpy     #INS_RESTART+1  ; counter = 23..34
478                 bcc     norestart       ; counter = 25..36
479
480                 lda     #$00            ; counter = 27..38
481                 sta     $d405,x         ; counter = 29..40
482                 lda     #$01            ; counter = 34..45
483                 sta     $d404,x         ; counter = 36..47
484
485                 ; Hard-restart 1
486                 ; Gate is enabled with adsr=0000
487                 ; and the correct adsr is set immediately afterwards.
488                 ; Hard-restart 2
489                 ; Switch to rate 0 with counter > 32
490                 ; to trigger second wraparound.
491                 ; Decay rate bug is avoided for both cases.
492 norestart
493                 lda     ins_ad-1,y
494                 sta     $d405,x
495                 lda     ins_sr-1,y
496                 sta     $d406,x
497 ins_done
498                 lda     #0
499                 sta     v_pendfx,x
500                 sta     v_pendins,x
501
502                 txa
503                 sbx     #7
504                 bpl     vloop
505                 .)
506         ;2 + 3 * 128 - 1 = 385
507
508                 lda     'zp_tempo
509                 sta     'zp_master
510 m_groove        = * + 1
```

```
511                     eor     #0
512                     sta     'zp_tempo
513
514                     lda     #<prepare1
515                     sta     preparejmp+1
516                     jmp     everyframe
517
518     ; ===========================================================================
519     ; Unpack more track data for voice x/7 (0-2).
520     ; 37 + 34 + 7 * 18 - 1 + 18 = 214 (read 7 bytes)
521     ; 38 + 55 + 229 - 1 + 15 = 336 (copy 10 notes)
522     ; Copying 10 notes would require 10 * 24 = 240 cycles.
523     ; But the cruncher ensures that the copy loop needs at most 229 cycles.
524     ; ===========================================================================
525
526     stopstream
527                     ldx     'zp_trwpos
528                     lda     #$c0            ; keep reading $c0 after end of song
529                     ldy     #1
530                     bne     poststop        ; always
531
532     unpackvoice
533                     .(
534                     lda     'zp_bufs+1,x
535                     sta     m_buf2+2
536                     sta     m_buf3+2
537
538                     ; time to unpack the next piece of compressed data?
539
540                     lda     v_trwpos,x
541                     cmp     'zp_bufs,x      ; writepos - readpos = bytes_in_buf
542                     bmi     postunpack      ; at least 128 bytes in buf, hold the flow
543
544                     sta     'zp_trwpos
545
546                     ; control byte is tttttnnn
547                     ; if t = 0, read n literal bytes
548                     ; if t > 0, copy n + 3 bytes with transpose t - 16, offset follows
549                     ; t = 0, n = 0 indicates stream end
550
551                     ldy     #0
552                     lax     (zp_inptr),y
553                     and     #$f8
554                     bne     copy
555             ; 37
556
557                     ; literal
558
559                     lda     m_buf2+2
560                     sta     m_buf1+2
561
562                     txa
563                     beq     stopstream
564
565                     tay
566
567                     eor     #$ff    ; 01 -> fe, 02 -> fd...
568                     clc
569                     adc     'zp_inptr
570                     sta     'zp_inptr
571                     bcs     noc1
572
573                     dec     'zp_inptr+1
574     noc1
575                     ldx     'zp_trwpos
576             ; 34
577     litloop
578                     lda     (zp_inptr),y
579     +poststop
```

```
580   +m_buf1
581                   sta     !0,x
582                   inx
583                   dey
584                   bne     litloop
585           ; 18 * 7 - 1
586
587                   txa
588                   jmp     postliteral
589   copy
590                   lsr
591                   lsr
592                   ;clc
593                   sbc     #$20-1
594                   sta     m_transp
595                   lda     #$07
596                   sbx     #$fd    ; x becomes number of bytes to copy
597
598                   lda     'zp_inptr
599                   ;clc
600                   sbc     #2-1
601                   sta     'zp_inptr
602                   bcs     noc2
603
604                   dec     'zp_inptr+1
605   noc2
606                   txa
607                   clc
608                   adc     'zp_trwpos
609                   sta     m_copyend
610
611                   iny
612   #if REPEAT
613                   clc
614                   adc     (zp_inptr),y
615                   tax
616   #else
617                   lax     (zp_inptr),y
618   #endif
619                   ldy     'zp_trwpos
620           ; 51 + 4 (repeat)
621   copyloop
622   +m_buf2
623                   lda     !0,x
624                   bmi     notransp
625
626                   clc
627   m_transp        = * + 1
628                   adc     #0
629   notransp
630   +m_buf3
631                   sta     !0,y
632                   inx
633                   iny
634   m_copyend       = * + 1
635                   cpy     #0
636                   bne     copyloop
637
638                   tya
639   postliteral
640                   ldx     'zp_master
641                   sta     v_trwpos,x
642                   .)
643   postunpack
644                   ldx     #14
645   preparejmp
646                   jmp     prepare1        ; opcode replaced with rts during init
647
648   ; ================================================================================
```

```
649   ; Data
650   ; ======================================================================
651
652   v_pwidth
653                   .byt    0
654   v_trwpos
655                   .byt    0
656   v_pendnote
657                   .byt    0
658   v_pendfx
659                   .byt    0
660   v_pendins
661                   .byt    0
662   v_wavemask
663                   .byt    $fe
664   v_trtimer
665                   .byt    $ff
666
667                   .byt    0,0,0,0,0,$fe,$ff
668                   .byt    0,0,0,0,0,$fe,$ff
669   v_fxpos
670                   .byt    0
671   v_currfx
672                   .byt    0
673   v_currins
674                   .byt    0
675   v_basepitch
676                   .byt    0
677   v_wavepos
678                   .byt    0
679                   .byt    0,0
680                   .dsb    7,0
681                   .dsb    5,0
682
683                   .dsb    (playorg + $400 - 207 - *), $ee
684   freq_msb
685                   .byt    $00,$00,$00,$00,$00,$00,$00,$00,$00,$00,$00,$00,$00,$01
686                   .byt    $01,$01,$01,$01,$01,$01,$01,$01,$01,$02,$02,$02,$02,$02
687                   .byt    $02,$02,$03,$03,$03,$03,$04,$04,$04,$04,$05,$05,$05,$06,$06
688                   .byt    $06,$07,$07,$08,$08,$09,$09,$0a,$0a,$0b,$0c,$0d,$0d,$0e,$0f,$10
689                   .byt    $11,$12,$13,$14,$15,$17,$18,$1a,$1b,$1d,$1f,$20,$22,$24,$27,$29
690                   .byt    $2b,$2e,$31,$34,$37,$3a,$3e,$41,$45,$49,$4e,$52,$57,$5c,$62,$68
691
692                   ; tables overlap with 15 bytes
693   freq_lsb
694                   .byt    $6e,$75,$7c,$83,$8b,$93,$9c,$a5,$af,$b9,$c4,$d0,$dd,$ea,$f8,$07
695                   .byt    $16,$27,$39,$4b,$5f,$74,$8a,$a1,$ba,$d4,$f0,$0e,$2d,$4e,$71,$96
696                   .byt    $be,$e7,$14,$42,$74,$a9,$e0,$1b,$5a,$9c,$e2,$2d,$7b,$cf,$27,$85
697                   .byt    $e8,$51,$c1,$37,$b4,$38,$c4,$59,$f7,$9d,$4e,$0a,$d0,$a2,$81,$6d
698                   .byt    $67,$70,$89,$b2,$ed,$3b,$9c,$13,$a0,$45,$02,$da,$ce,$e0,$11,$64
699                   .byt    $da,$76,$39,$26,$40,$89,$04,$b4,$9c,$c0,$23,$c8,$b4,$eb,$72,$4c
700                   .byt    $80,$12,$08,$68,$39,$80,$45,$90,$68,$d6,$e3,$99,$00,$24,$10
701
702                   ; this becomes page-aligned
703   pwprepare
704                   .byt    $8f,$7f,$6f,$5f,$4f,$3f,$2f,$1f,$0f,$fe,$ee,$de,$ce,$be,$ae,$9e
705                   .byt    $8e,$7e,$6e,$6e,$5e,$4e,$3e,$2e,$1e,$0e,$fd,$ed,$dd,$cd,$bd,$ad
706                   .byt    $9d,$8d,$7d,$6d,$5d,$5d,$4d,$3d,$2d,$1d,$0d,$fc,$ec,$dc,$cc,$bc
707                   .byt    $ac,$9c,$8c,$7c,$6c,$5c,$4c,$4c,$3c,$2c,$1c,$0c,$fb,$eb,$db,$cb
708                   .byt    $bb,$ab,$9b,$8b,$7b,$6b,$5b,$4b,$3b,$3b,$2b,$1b,$0b,$fa,$ea,$da
709                   .byt    $ca,$ba,$aa,$9a,$8a,$7a,$6a,$5a,$4a,$3a,$2a,$2a,$1a,$0a,$f9,$e9
710                   .byt    $d9,$c9,$b9,$a9,$99,$89,$79,$69,$59,$49,$39,$29,$19,$19,$09,$f8
711                   .byt    $e8,$d8,$c8,$b8,$a8,$98,$88,$78,$68,$58,$48,$38,$28,$18,$08,$08
712                   .byt    $08,$08,$18,$28,$38,$48,$58,$68,$78,$88,$98,$a8,$b8,$c8,$d8,$e8
713                   .byt    $f8,$09,$19,$19,$29,$39,$49,$59,$69,$79,$89,$99,$a9,$b9,$c9,$d9
714                   .byt    $e9,$f9,$0a,$1a,$2a,$2a,$3a,$4a,$5a,$6a,$7a,$8a,$9a,$aa,$ba,$ca
715                   .byt    $da,$ea,$fa,$0b,$1b,$2b,$3b,$3b,$4b,$5b,$6b,$7b,$8b,$9b,$ab,$bb
716                   .byt    $cb,$db,$eb,$fb,$0c,$1c,$2c,$3c,$4c,$4c,$5c,$6c,$7c,$8c,$9c,$ac
717                   .byt    $bc,$cc,$dc,$ec,$fc,$0d,$1d,$2d,$3d,$4d,$5d,$5d,$6d,$7d,$8d,$9d
```

```
718                     .byt    $ad,$bd,$cd,$dd,$ed,$fd,$0e,$1e,$2e,$3e,$4e,$5e,$6e,$6e,$7e,$8e
719                     .byt    $9e,$ae,$be,$ce,$de,$ee,$fe,$0f,$1f,$2f,$3f,$4f,$5f,$6f,$7f,$8f
720
721     #if REPEAT
722                     .seg    seg_rinit
723     #else
724                     .seg    seg_init
725     #endif
726
727     initroutine
728                     .(
729                     lda     #<streamstart
730                     sta     'zp_inptr
731                     lda     #>streamstart
732                     sta     'zp_inptr+1
733
734                     lda     #<prepare1
735                     sta     preparejmp+1
736                     lda     #0
737                     sta     'zp_extsync
738                     sta     'zp_pendoob
739                     sta     'zp_filtpos
740
741                     ldx     #$18
742     clr
743                     sta     $d400,x
744                     dex
745                     bpl     clr
746
747                     lda     #$80
748                     sta     m_cutoff
749
750                     ldy     #>(unpackbufs+$200)
751                     ldx     #14
752     vloop1
753                     sty     'zp_bufs+1,x
754                     dey
755                     lda     #0
756                     sta     'zp_bufs,x
757                     sta     v_trwpos,x
758                     sta     v_pendfx,x
759                     sta     v_pendins,x
760                     lda     #$ff
761                     sta     v_trtimer,x
762                     sbx     #7
763                     bpl     vloop1
764
765                     lda     #$60
766                     sta     preparejmp
767                     ldx     #7
768                     jsr     stx_unpackvoice
769                     jsr     playroutine
770                     lda     #$4c
771                     sta     preparejmp
772
773                     lda     #3*7
774                     sta     'zp_master
775
776                     rts
777                     .)
```