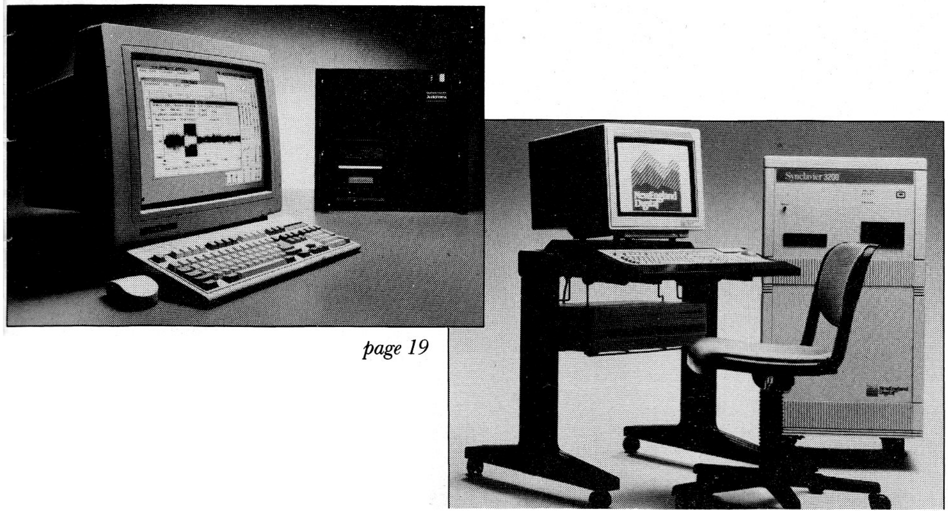
Electronic Musician

AN ACT III PUBLICATION
MARCH 1989 VOL. 5, NO. 3

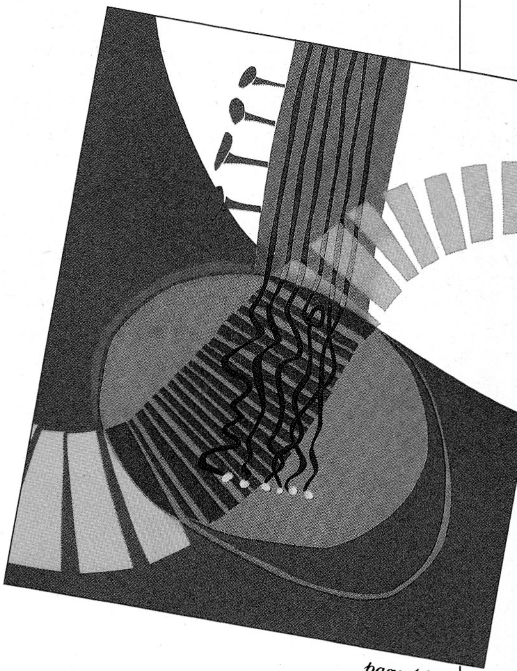
features

applications

do-it-yourself



page 19



page 44

reviews

departments

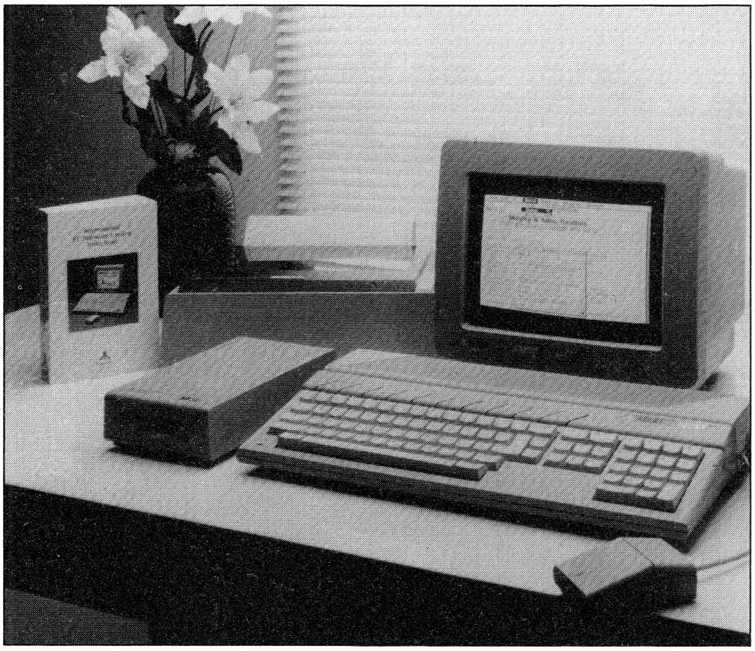
Cover: Equipment photo by David Bishop. Clouds photo ©Mark Stephenson/West Light.

Electronic Musician® is published at 6400 Hollis St. #12, Emeryville, CA 94608, and is © 1989 by NBB Acquisitions, Inc. This is Volume 5, Number 3, March 1989. **Electronic Musician** (ISSN: 0884-4720) is published monthly. Second Class postage paid at Oakland, CA, and additional mailing offices. All rights reserved. This publication may not be reproduced or quoted in whole or in part by any means, printed or electronic, without the written permission of the publishers.

THE MIDI MUSIC BOX: More Cheap Thrills for the Atari ST

In our continuing quest to deliver something for nothing, we present a new algorithmic composition program.

By David Snow



The MIDI Music Box algorithmic composition software for the Atari ST is essentially a bare-bones sequencer that generates its own tracks and is sure to provide hours of mind-numbing diversion. More important, the program illustrates some interesting programming techniques that you can incorporate into your own programs.

The Music Box requires no input parameters. Pressing the "G" key will generate four tracks, each 675 notes long, that draw their rhythm and pitch material from predetermined source sets. Pressing "P" will play the four monophonic tracks on MIDI channels 1 to 4, which is dandy if you have a CZ-101 or other multitimbral synth. During playback the ST's sound chip provides a metronome click on every beat. You can change playback tempo from the default value of 120 beats per minute by pressing "T" and entering a new value.

The program requires two files: a "shell" written in LDW BASIC (**Listing 1**) that initializes the program and manages the user interface, and a collection of assembly language routines called MIDIMUSE.BIN (**Listing 2**) that do the dirty work. The GENST.PRG public domain editor/assembler was used to assemble the latter file. (*Note: LDW BASIC is a compiled language that is somewhat different than ST BASIC, the language supplied by Atari. See the sidebar for details on converting the program to ST BASIC. You can assemble MIDIMUSE.S with just about any 68000 assembler, though you may have to make a few slight changes, especially in the comments and labels.—Ed.*)

The Music Box records its track data in four arrays (track_a, track_b, track_c, track_d), each array being 2,700 words (5,400 bytes) long. Each MIDI event requires four bytes, and each note in the track requires two MIDI events (a note on and a note off), giving a total of 675 notes per track (5,400÷[4+4]=675). The data of each MIDI event is structured as follows: timestamp, two bytes; pitch, one byte; velocity, one byte.

The timestamp is a word-long value that indicates the time of the event, relative to the beginning of the sequence. The pitch and velocity bytes are normal MIDI data. The only difference between note on and note off events in Music Box is the velocity value: 64 for a note on, 0 for a note off. If a note on is not followed by a note off, the note will hang.

Every track is terminated with an end-of-track marker to which the arbitrary value of 253 is assigned. This number is placed in the third byte of the last MIDI event of each track, just after the last timestamp.

That's it. This rigid four-byte structure doesn't allow for niceties like pitch bend, but that was a tradeoff to keep things simple.

GETTING WITH THE PROGRAM

The eight source rhythm patterns from which the program constructs each track are encoded in data statements under the LOAD RHYTHM PATTERNS heading in listing 1. Each pattern consists of ten numbers, organized as five pairs of two numbers; the first in each pair represents the position of each note in the pattern, the second represents the duration of the note. Each beat is subdivided into 24 pulses. Therefore, in a pattern of four quarter notes, the first would occur on pulse 0, the second on pulse 24, the third on pulse 48, and the last on pulse 72. Since each quarter note is 24 pulses long, the pattern would be represented as: 0, 24, 24, 24, 48, 24, 72, 24. To show where the next note will fall, the pattern must end with the next pulse position, followed by a duration value of 0, in this case, 96, 0.

Using this formula, it's easy to read the patterns encoded in this set of data statements: pattern 1 is a single eighth note (the durations of the notes in each pattern are truncated slightly for the sake of articulation), pattern 2 is two sixteenths, pattern 3 is a sixteenth followed by a dotted eighth rest, pattern 4 is a quarter note, pattern 5 is two eighth notes, pattern 6 is a dotted eighth and a sixteenth, pattern 7 is a dotted quarter note, and pattern 8 is a half note. If you don't like the patterns generated by the program, change the data statements accordingly and (if you're using LDW BASIC) recompile it; just keep each pattern to four notes or less.

To generate a rhythmic track, the program first resets a timestamp counter to 1. It then generates a random number from zero to seven, corresponding to one of the eight possible patterns. The program looks at the first position value of the selected pattern (which is always zero), adds that to the timestamp counter, and places the sum in the first two bytes of the selected track. It then clears the pitch value byte and places a value of 64 in the velocity byte. That makes four bytes so far, the first MIDI event of the track. The duration value for the rhythmic pattern is

CONVERTING TO ST BASIC

The program listing presented here was written in LDW BASIC (manufactured by Logical Design Works), a compiled version of BASIC that runs on the ST. This is a fast language, suitable for creating stand-alone programs that can be executed directly from the desktop, but it does have one disadvantage over ST BASIC: it's not included with the computer, so you'll need to buy a copy in order to run this type of program. However, if ST BASIC is all you have, all is not lost; with a few modifications, the Music Box will run in ST BASIC, though not as quickly. The several changes involved are simple: in addition to changes to the BASIC program itself, you'll need to make a few changes to the assembly language portion of the program. These changes are listed below.

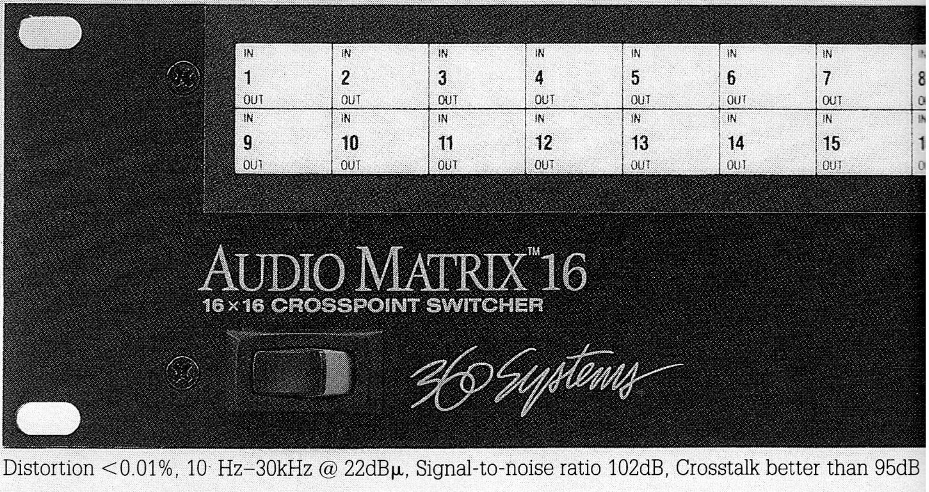
1. ST BASIC does not allow the use of the underscore (_) character in variable names or labels, so replace each occurrence of the underscore with a period. For example, track_a becomes track.a, set_tempo becomes set.tempo, etc.

2. Change each occurrence of the % symbol to #. For example, track% becomes track#.

3. Delete the word "static" from the DIM statements in lines 20-60.

4. Change the following lines accordingly:

```
10    defint a-z
60    dim muse.bin(400)
260   for index#=0 to 11
270   read pitch: poke index#, pitch
310   for track=0 to 3: def seg = track#(track):poke 2,253:next
340   print "PRESS 'T' TO ENTER TEMPO (BEATS PER MINUTE): "
430   if key=81 or key=113 then poke systab+24,0: end
460   gotoxy 0,6: print "WORKING"
510   gotoxy 0,6: print spc(7)
540   gotoxy 0,2: print spc(5)
550   gotoxy 0,2: input " ", tempo
590   gotoxy 0,6: print "PLAYING"
```



then added to the last timestamp, and the sum is placed in the next timestamp location in the track. The next pitch byte is cleared and the value of "0" placed in the velocity byte. That makes another four bytes, the second MIDI event. The next position value of the rhythmic pattern is added to the timestamp counter and the sum placed in the next timestamp. The program then looks at the next duration value. If it is any number greater than zero, the note on/note off data process just described is repeated. If it equals zero, the pattern is ended. A new random number is generated, another pattern selected, and so on, until it gets to the end of the track. Then the last timestamp is put in, the end-of-track marker tacked on (253), and the routine ends. The routine to generate pitches is simpler, randomly selecting pitches from the pitch_bufr array and stuffing them into the appropriate slots in each MIDI event until an end-of-track marker is encountered. The source pitch set is encoded in the data statement under LOAD PITCH SET in Listing 1 and can also be changed to suit your preferences.

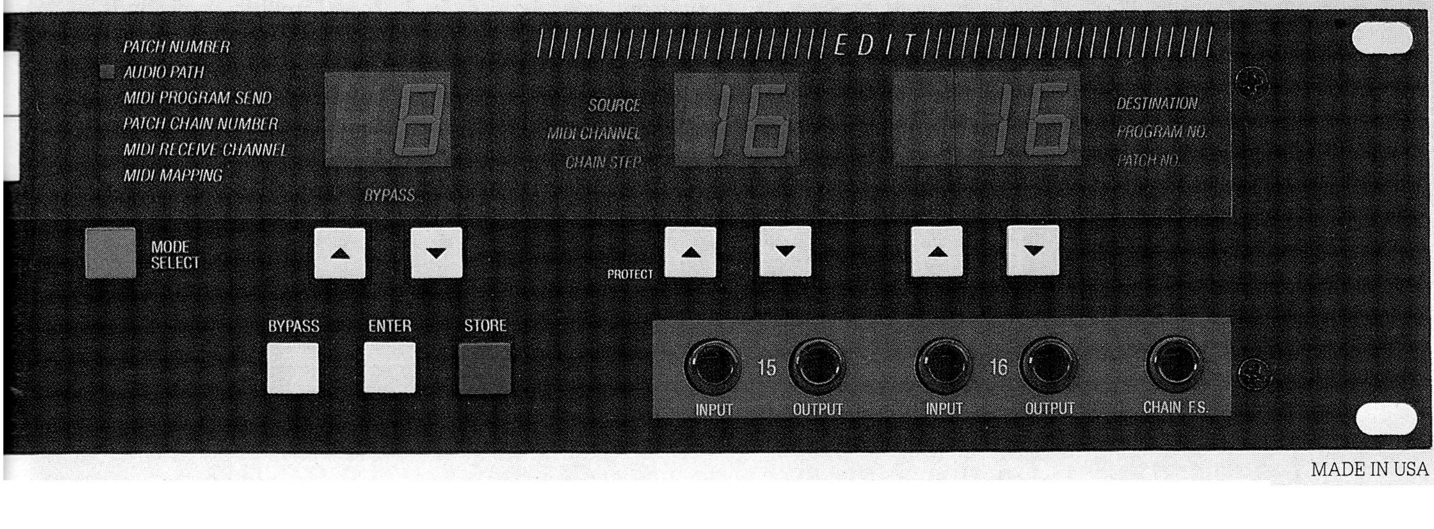
```
600   gotoxy 0,7: print "(HIT ANT KEY TO CONTINUE)"
620   gotoxy 0,6: print spc(7)
630   gotoxy 0,7: print spc(21)
```

5. Add the following lines to the program:

```
5     on error go to 1000
45    track.a(0)=0: track.b(0)=0: track.c(0)=0: track.d(0) = 0
255   def seg = pitch.bufr#
325   fullw 2: clearw 2: gotoxy 0,0
345   print tempo
365   def seg = 0: poke systab+24,1
535   poke systab+24,0
565   poke systab+24,1
1000  poke systab+24,0
```

You'll also need to change some sections of the assembly language program to read as follows:

```
START   movem.l do-d7/ao-a6,-(a7)   SAVE REGISTERS
        movea.l 66(a7),a6     GET PARAMETER POINTER A6 FROM STACK
        .
        .
RHYTHM  move.l 4(a6),a5     GET TRACK POINTER A5
        move.l 8(a6),a4     GET RHYTHM-BUFFER POINTER A4
        .
        .
PITCH   move.l 4(a6),a5     GET TRACK POINTER A5
        move.l 8(a6),a4     GET BUFFER POINTER A4
        .
        .
BASIC   movem.l (a7)+,d0-d7/a0-a6   RESTORE REGISTERS
        rts
        .
        .
PLAY    movea.l a6,a2
        move.l 4(a2),a6      GET TRACK POINTER 'A'
        move.l 8(a2),a5      GET TRACK POINTER 'B'
        move.l 12(a2),a4     GET TRACK POINTER 'C'
        move.l 16(a2),a3     GET TRACK POINTER 'D'
        move.l 20(a2),d5     GET TEMPO VALUE D5
```



MORE UGLY DETAILS

The ST operating system has a few built-in MIDI functions that simplify programming. These subroutines are part of the Basic Input/Output System (BIOS) and extended BIOS (XBIOS) and are accessible to the assembly language programmer through instructions called "traps" that direct program execution to these subroutines. Music Box routines called in this fashion are used for outputting MIDI data and for providing the metronome click.

The ST also simplifies timekeeping duties with a Motorola MC68901 Multi-Function Peripheral (MFP) chip, which has four programmable interrupt-generating timers, only three of which are used by the operating system. The routine in Listing 2 identified as MFPTIM programs the unused timer to generate an interrupt 4,000 times per second. When the interrupt is generated, the program stops whatever it's doing, jumps to the routine labeled TIMER, increments the variable

> **The Music Box is a sequencer that generates its own tracks.**

Electronic Musician MIDI Music Box for Atari ST (Shell)

```
10   defwrd a-z
20   dim static track a(2700): dim static track b(2700)
30   dim static track_c(2700): dim static track_d(2700)
40   dim static rhythm bufr(10,10)
50   dim static pitch bufr(6)
60   dim static muse bin(350)
70   track%(0)=varptr(track_a(0)): track%(1)=varptr(track_b(0))
80   track%(2)=varptr(track c(0)): track%(3)=varptr(track_d(0))
90   rhythm bufr%=varptr(rhythm bufr(0,0))
100  pitch bufr%=varptr(pitch bufr(0))
110  muse_bin%=varptr(muse_bin(0))
120  bload "midimuse.bin",muse_bin%
130  ' LOAD RHYTHM PATTERNS
140  for phrase=0 to 7: for element=0 to 9
150       read rhythm_bufr(phrase,element)
160  next element: next phrase
170       data 0,10,12,0,0,0,0,0,0,0
180       data 0,4,6,4,12,0,0,0,0,0
190       data 0,6,24,0,0,0,0,0,0,0
200       data 0,20,24,0,0,0,0,0,0,0
210       data 0,10,12,10,24,0,0,0,0,0
220       data 0,16,18,4,24,0,0,0,0,0
230       data 0,32,36,0,0,0,0,0,0,0
240       data 0,46,48,0,0,0,0,0,0,0
250  'LOAD PITCH SET
260  for index=0 to 11
270       read pitch: poke_b pitch_bufr%+index,pitch
280  next
290       data 43,50,53,55,59,60,62,65,67,68,70,71
300  ' INITIALIZE TRACKS WITH END-OF-TRACK MARKERS
310  for track=0 to 3: poke_b track%(track)+2,253: next
320  tempo=120: cycle_count=107
330  print "PRESS 'G' TO GENERATE TRACKS 1 TO 4"
340  print "PRESS 'T' TO ENTER TEMPO (BEATS PER MINUTE): ";tempo
350  print "PRESS 'P' TO PLAY"
360  print "PRESS 'Q' TO QUIT"
370  while 1
380       while inp(-2)¼½-1: wend
390       key=inp(2)
400       if key=71 or key=103 then gosub generate
410       if key=84 or key=116 then gosub set_tempo
420       if key=80 or key=112 then gosub playback
430       if key=81 or key=113 then end
440  wend
450  generate:
460       gotoxy 36,12: print "WORKING"
470       for track=0 to 3
480            call muse_bin%(1,track%(track),rhythm_bufr%)
490            call muse_bin%(2,track%(track),pitch_bufr%)
500       next
510       gotoxy 36,12: print spc(7)
520       return
530  set_tempo:
540       gotoxy 45,1: print spc(5)
550       gotoxy 45,1: input " ",tempo
560       cycle_count=12840 tempo
570       return
580  playback:
590       gotoxy 36,12: print "PLAYING"
600       gotoxy 29,13: print "(HIT ANY KEY TO QUIT)"
610       call muse_bin%(3,track%(0),track%(1),track%(2),track%(3),cycle_count)
620       gotoxy 36,12: print spc(7)
630       gotoxy 29,13: print spc(21)
640       return
```

LISTING 1: LDW BASIC Source Code (MIDIMUSE.BAS). Copyright 1988, David Snow.

Electronic Musician MIDI Music Box for Atari ST (Assembly Language Routines)

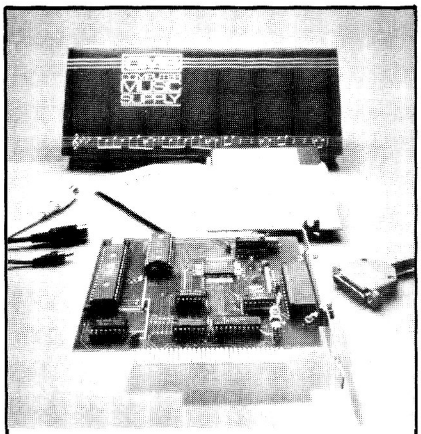
```
* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
*  MIDI MUSIC BOX ASSEMBLY LANGUAGE SUBROUTINES
* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
START   movea.l 6(a7),a6     GET PARAMETER POINTER A6 FROM STACK
        move.l 4(a6),d7      GET ROUTINE INDEX D7
        cmpi.b #1,d7         BRANCH TO INDEXED ROUTINE
        beq RHYTHM
        cmpi.b #2,d7
        beq PITCH
        cmpi.b #3,d7
        beq PLAY
* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
* GENERATE RHYTHMIC SEQUENCE
* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
RHYTHM  move.l 12(a6),a5     GET TRACK POINTER A5
        move.l 20(a6),a4     GET RHYTHM-BUFFER POINTER A4
        move.l #5392,a6
        adda.l a5,a6         GET END-OF-TRACK POINTER A6
        move.l #1,d7         RESET TIMESTAMP COUNTER D7
MAKEIT  cmpa.l a5,a6         END OF TRACK?
        bls TRMSEQ
        cmpi.l #65535,d6     MAXIMUM TIMESTAMP VALUE?
        bhi QUITIT
        move.w #17,-(a7)     GENERATE RANDOM NUMBER DO
        trap #14
        addq.l #2,a7
        andi.l #7,d0         LIMIT TO 3 BITS (D0¼=7)
        mulu.w #22,d0        CALCULATE PATTERN ADDRESS OFFSET
        movea.l d0,a3
        adda.l a4,a3         GET RHYTHM POINTER A3
UPDATE  move.w d7,d6         GET NOTE-ON TIMESTAMP D6
        add.w (a3),d6        UPDATE NOTE-ON TIMESTAMP
        cmpi.w #0,2(a3)      END OF RHYTHM PATTERN?
        bne STAMP
        add.w (a3),d7        UPDATE TIMESTAMP COUNTER
        bra MAKEIT           NEXT PATTERN
STAMP   move.w d6,(a5)+      LOAD NOTE-ON TIMESTAMP TO TRACK
        move.b #0,(a5)+      sLI
                                (RDATA

        move.b #64,(a5)+     LOAD VELOCITY TO TRACK (NOTE ON)
        addq.l #2,a3         POINT TO NEXT RHYTHMIC ELEMENT
        move.w d6,d5         GET NOTE-OFF TIMESTAMP D5
        add.w (a3),d5        UPDATE NOTE-OFF TIMESTAMP
        move.w d5,(a5)+      LOAD NOTE-OFF TIMESTAMP TO TRACK
        clr.w (a5)+          CLEAR PITCH AND VELOCITY
        addq.l #2,a3         POINT TO NEXT RHYTHMIC ELEMENT
        bra UPDATE
QUITIT  move.w #65535,d6
TRMSEQ  move.w d6,(a5)+
        move.b #253,(a5)     APPEND END-OF-TRACK MARKER
        rts
* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
* GENERATE PITCH SEQUENCE
* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
PITCH   move.l 12(a6),a5     GET TRACK POINTER A5
        move.l 20(a6),a4     GET BUFFER POINTER A4
        move.l #11,d6        GET NUMBER OF NOTES-IN-BUFFER D6
        movea.l #5400,a6
        adda.l a5,a6         GET END-OF-TRACK POINTER A6
DOLOOP  cmpa.l a5,a6         END OF TRACK?
        bls BASIC
        addq.l #2,a5         SKIP TIMESTAMP
        cmpi.b #253,(a5)     END OF SEQUENCE?
        beq BASIC
        move.w #17,-(a7)     GENERATE RANDOM NUMBER DO
        trap #14
        addq.l #2,a7
        andi.l #$000F,d0     LIMIT TO 4 BITS
TRYIT   cmp.w d6,d0          D0¼=D6?
        bls KEEPIT
        lsr.w #1,d0          IF NOT, SHIFT DO RIGHT
        bra TRYIT
KEEPIT  movea.l a4,a3
        adda.w d0,a3
        move.b (a3),(a5)     GET PITCH FROM BUFFER (NOTE ON)
```

LISTING 2: Assembly Language Source Code (MIDIMUSE.S). Copyright 1988, David Snow.

```
        addq.l #4,a5
        move.b (a3),(a5)     GET PITCH FROM BUFFER (NOTE OFF)
        addq.l #2,a5         JUMP TO NEXT TIMESTAMP
        bra DOLOOP
BASIC   rts
* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
* PLAYBACK
* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
PLAY    movea.l a6,a2
        move.l 12(a2),a6     GET TRACK POINTER 'A'
        move.l 20(a2),a5     GET TRACK POINTER 'B'
        move.l 28(a2),a4     GET TRACK POINTER 'C'
        move.l 36(a2),a3     GET TRACK POINTER 'D'
        move.l 44(a2),d5     GET TEMPO VALUE D5
        move.w #4,d0
        bsr MFPTIM           START MFP TIMER
        clr.w d3             RESET TIME COUNTER D3
CLOCK   pea SOUND(PC)        CLICK METRONOME
        move.w #32,-(a7)
        trap #14
        addq.l #6,a7
        move.w #24,d2        RESET PULSE COUNTER D2
BEAT    lea COUNT(PC),a2
        clr.w (a2)           RESET CYCLE COUNTER (A2)
        addq.w #1,d3         INCREMENT TIME COUNTER
        subq.w #1,d2         DECREMENT PULSE COUNTER D2
        tst d2
        beq CLOCK
RECYCL  cmp.w (a2),d5        HAS CYCLE COUNTER EXPIRED?
        bls BEAT
        movem.l d1-d2/a2,-(a7)
        move.w #2,-(a7)      CHECK FOR KEY PRESS (EXIT PLAYBACK)
        move.w #1,-(a7)
        trap #13
        addq.l #4,a7
        movem.l (a7)+,d1-d2/a2
        tst d0
        bne RETURN
        cmp.b #253,2(a6)     END OF TRACK A?
        beq TRACKB
        move.w (a6),d0       TIMESTAMP 'A'=TIME COUNTER VALUE?
        cmp.w d3,d0
        bhi TRACKB
        addq.l #2,a6
OUTPUTA move.w #144,d0       OUPUT NOTE-ON COMMAND
        bsr MIDOUT
        move.b (a6)+,d0      OUTPUT PITCH DATA
        bsr MIDOUT
        move.b (a6)+,d0      OUTPUT VELOCITY DATA
        bsr MIDOUT
TRACKB  cmp.b #253,2(a5)     END OF TRACK B?
        beq TRACKC
        move.w (a5),d0       TIMESTAMP 'B'=TIME COUNTER VALUE?
        cmp.w d3,d0
        bhi TRACKC
        addq.l #2,a5
OUTPUTB move.w #145,d0       NOTE-ON COMMAND
        bsr MIDOUT
        move.b (a5)+,d0      PITCH
        bsr MIDOUT
        move.b (a5)+,d0      VELOCITY
        bsr MIDOUT
TRACKC  cmp.b #253,2(a4)     END OF TRACK C?
        beq TRACKD
        move.w (a4),d0       TIMESTAMP 'C'=TIME COUNTER VALUE?
        cmp.w d3,d0
        bhi TRACKD
        addq.l #2,a4
OUTPUTC move.w #146.d0       NOTE-ON COMMAND
        bsr MIDOUT
        move.b (a4)+,d0      PITCH
```

LISTING, continued on page 66



LISTING, from page 65

```
        bsr MIDOUT
        move.b (a4)+,d0      VELOCITY
        bsr MIDOUT
TRACKD  cmp.b #253,2(a3)     END OF TRACK D?
        beq RECYCL
        move.w (a3),d0       TIMESTAMP 'D'=TIME COUNTER VALUE?
        cmp.w d3,d0
        bhi RECYCL
        addq.l #2,a3
OUTPUTD move.w #147,d0       NOTE-ON COMMAND
        bsr MIDOUT
        move.b (a3)+,d0      PITCH
        bsr MIDOUT
        move.b (a3)+,d0      VELOCITY
        bsr MIDOUT
        bra RECYCL
RETURN  move.w #2,-(a7)      EMPTY KEYBOARD BUFFER
        move.w #2,-(a7)
        trap #13
        addq.l #4,a7
        move.w #0,d0
        bsr MFPTIM           STOP TIMER
        bsr SILENT           ALL NOTES OFF
        rts
*
*  MIDI DATA OUPUT SUBROUTINE
*
MIDOUT  movem.l d1-d2/a2,-(a7)
        move.w d0,-(a7)
        move.w #3,-(a7)
        move.w #3,-(a7)
        trap #13
        addq.l #6,a7
        movem.l (a7)+,d1-d2/a2
        rts
*
* ALL NOTES OFF SUBROUTINE
*
SILENT  lea ALLOFF(PC),a6
        move.w #5,d4
        clr.w d0
OFF     move.b (a6)+,d0
        bsr MIDOUT
        dbra d4,OFF
        rts
*
*   START/STOP MFP TIMER SUBROUTINE
*
MFPTIM  pea TIMER(PC)        GET INTERRUPT ROUTINE VECTOR
        move.w #10,-(a7)     COUNT DOWN FROM 10
        move.w d0,-(a7)      START/STOP TIMER
        move.w #0,-(a7)      SELECT TIMER A
        move.w #31.-(a7)     XBTIMER ROUTINE
        trap #14
        add.l #12,a7
        rts
*
*   TIMER INTERRUPT ROUTINE
*
TIMER   movem.l d0-d7/a0-a6,-(a7)
        lea COUNT(PC),a5
        addq.w #1,(a5)       INCREMENT PULSE COUNTER
        bclr #5,$FFFFFA0F    CLEAR INTERRUPT SERVICE BIT
        movem.l (a7)+,d0-d7/a0-a6
        rte
*
SOUND   dc.w $005A,$0100,$073E,$0810,$0B00,$0C09,$0D00,$FF00
ALLOFF  dc.w $B07A,$00B0,$7A7F
COUNT   ds.w 1
 end
```

The ST operating system has a few built-in MIDI functions that simplify programming.

COUNT, then jumps back to where it was before the interrupt. The playback routine continually checks COUNT to see whether its value matches the cycle_count variable that was passed from the shell. If it does, the time span of one pulse has expired, COUNT is reset to 0, and both a pulse counter and a timestamp counter are incremented. When the pulse counter reaches 24, one beat has expired, the pulse counter is reset to 0, and the metronome clicks. The timestamp counter is used by the playback routine to control the output of each MIDI event in each track.

All notes are turned off at the termination of the playback routine by outputting a data string labeled ALLOFF (see the end of Listing 2) through the MIDI port. This is not the genuine all notes off MIDI message, which the CZ-101 does not recognize, but rather the local control off/local control on commands which, for our purposes, have the same effect.

You can probably think of dozens of enhancements to the Music Box that will make it more powerful, such as real-time track recording, MIDI keyboard input of pitch sets, track-to-track copying, track inversion, reversal, augmentation, and diminution. As a matter of fact, these listings were extracted from Music Box, a mouse-oriented application for ST systems that features those capabilities and a few more. The source code and program files for Music Box (LDW BASIC version) are available to interested **EM** readers (for a limited time only) by sending $8 ($10 outside the U.S. for 1st Class mail delivery) to EM Bookshelf (see FYI page for more information).

David Snow *holds degrees in composition from Eastman and Yale and is the recipient of numerous commissions, awards, and grants, including some from BMI, ASCAP, and the National Endowment for the Arts. His music has been performed at the Kennedy Center, Carnegie Hall, and Tanglewood.*