

MELODY MAKER

by Dave Hassler
for the PAL-1 6502 computer
www.vanportmedia.com/PAL-1

A program to aid in creating music for
Jim Butterfield's MUSIC BOX
as presented in "The First Book of KIM," page 91

Melody Maker and its documentation (c) 2023 by David H. Hassler
Licensed for use under the MIT License

TABLE OF CONTENTS

1. Introduction
2. Operation: Input and Editing
3. Sound Output
4. Applications and Examples
5. Tables and Songs

I. INTRODUCTION

The PAL-1 is an amazing hobby computer. At 2K of ROM for the system and monitor, plus 5K of user RAM, it's possible to hold the entire 6502 system in your head, just as you could in 1976 with the original KIM-1. And one of the grand gurus of the 6502 was, undoubtedly, Jim Butterfield. Jim's MUSIC BOX program plays notes from a user-generated file, just like a player piano: MUSIC BOX is the piano, and the data file is the piano roll.

The program is genius. It uses the PB7 line on a 6530 RRIOT chip – the same line used by the KIM-1's cassette tape output routine – to pump out tunes. He notes in "The First Book of KIM" that one can record tunes directly into the tape recorder, or send the output to a little amplifier. What this means for you, dear unexpanded PAL-1 owner, is that you can have music and sounds come out of your PAL-1 without having to buy or build a separate 6532 RIOT board to get all those Port A and Port B I/O lines: Liu Ganning – the PAL-1's creator – has provided us with PB7 on the PAL's expansion connector, since it's needed for cassette storage. But, as Jim knew way back when, there's certainly no reason why you can't crank out *other sounds* from that line.

The only problem with MUSIC BOX, as originally presented, is that you have to do a translation between the table of hexadecimal numbers that represent the notes and actual sheet music, plus remember to make control code changes when changing between, say, a quarter note and a half note, or when crossing into a different octave. Plus, there are *two sets* of note codes: one for eighth notes and one for everything else. Melody Maker was written a) as an aid to creating music files, and

b) to see if I could do it. I am by no means an “expert” at any of this (I was in non-technical areas most of my working life); it’s all part of my self-education in computer science and micro electronics, just as ham radio served as my introduction to basic electronics, AF circuits, and RF in general.

Melody Maker allows the user to look at a piece of sheet music, see a half note (F-sharp low in the treble clef (octave 2), for this example) and type `FS2 4` into it instead of `FC Ø4 CØ` into the KIM-1 monitor: no referencing note or control code tables on a piece of paper – see note, type note. Melody Maker covers almost all of the bass and treble clefs, from a low of A at the bottom of the bass clef (A2 – 110 Hz) to the A above the treble clef (A5 – 880 Hz), a range of three octaves.

Now, a word about the word “octave.” For purposes of this program, I am measuring octaves starting with A rather than C, the latter of which is traditional. The reasoning for this stems from the fact that Butterfield’s original note table starts at A and I wanted to keep the player’s internals as they were. So, when I say “the low A of Octave 1” (the lowest note MUSIC BOX can play), the note (AN1) to which I refer is typically called A2 in the music biz; middle C on a score is a note of 261.6 Hz, called C4 in the literature, but referred to here as “CN2”. I hope this isn’t confusing to pro musicians who might use Melody Maker, but I don’t think it’s a big problem for the rest of us. Just remember that Octave 1 is in the bass clef, 3 in the treble clef, and 2 is in between, for the most part.

Considering that he was working with 1975 8-bit tech and just 1 kilobyte or RAM, it’s pretty cool that Jim was able to get so much out of the machine with only 139 bytes of code. The note table he gives in “The First Book of KIM” (he was also a pianist) is pretty close to accurate from what I measured (I used a guitar tuner and fIDigi – a ham radio audio spectrum display software, among many other things). A few notes stood a little tweaking, mainly to make those not spot-on consistently sharp instead of having some notes flat. Most of Octave 2 and the bottom half of Octave 3 are pretty solid; from the A below middle C to the G at the top of the treble clef is decent – call it two octaves. Then, there are some clunkers, notes that just couldn’t be tweaked further without a complete rewrite of the player, and that’s above my pay grade. Those notes are:

High A – the highest note (AN4 here, A5 technically), quite sharp (by 11 Hz)

D#, 3rd octave - quite sharp

G#, 2nd octave - quite sharp

Most of the 1st octave – it’s all sharp to one degree or another

The timing algorithm in MUSIC BOX that creates bass notes is at fault here. I got it as close as I could; adjustments here would have really fouled up the second octave, which is almost dead-on-frequency.

The bottom half of the bass notes behave better than the top of the octave, but not a lot. If you stick to the treble clef and just below – octaves 2 and 3 – for the most part, your tunes will come out sounding really good. There’s a chart at the end of this document with all of the notes and their internal frequency codes. It’ll all make sense once you take a look at Jim’s source code.

As alluded earlier, this works with an unexpanded PAL-1 and a serial terminal emulator/TTY connection (RAM expansion is dandy, too). MUSIC BOX is certainly usable without TTY, as originally intended, but the whole point of Melody Maker is to remove at least the level of tedium in figuring out everything with paper and pencil, then hand-entering code byte-by-byte on the little hex keypad.

Finally, you’re going to need some way to amplify the signal coming out of PB7, and I have outlined several methods in Section III that you can use successfully.

Now, on with the show...

II. OPERATION

Part A: INPUT

Melody Maker loads at \$0DD0. The MOS Papertape file contains both the program and a copy of eWozLite, my customized version of the eWoz monitor for the unexpanded PAL-1. It's included for a couple of reasons: it's an order of magnitude better than the KIM-1 monitor (for most things), and it's super-handly for editing music data files – none of those “pesky PTP checksums” to get in the way! eWozLite starts at \$1220. First and foremost, make *sure* your caps lock key is on. Once the program is loaded, start it with the KIM-1 monitor G command at address \$0DD0 (or DDØR in eWozLite).

LIMITATIONS

Due to being written in a 1K RAM environment, MUSIC BOX can only work on data contained on a single page. A song's data cannot cross a page boundary. The lower down on a page you can start entry, the longer your song can be, up to a limit of 255 bytes. Melody Maker will display the address of the current note being entered so you can see if you're getting close to a page boundary. One way around this limit for longer songs is outlined in the ML section of the Applications chapter.

STARTING ADDRESS

A title and very brief instructions are shown, then you are asked for the starting address of the song. Choosing **FF** will begin entry at the default address of \$028C (MUSIC BOX loads at \$0200 unless relocated). The ability to choose the starting address is handy when you need a song longer than 116 bytes, if you're tucking away game sounds in the RIOT RAM, or you've relocated MUSIC BOX.

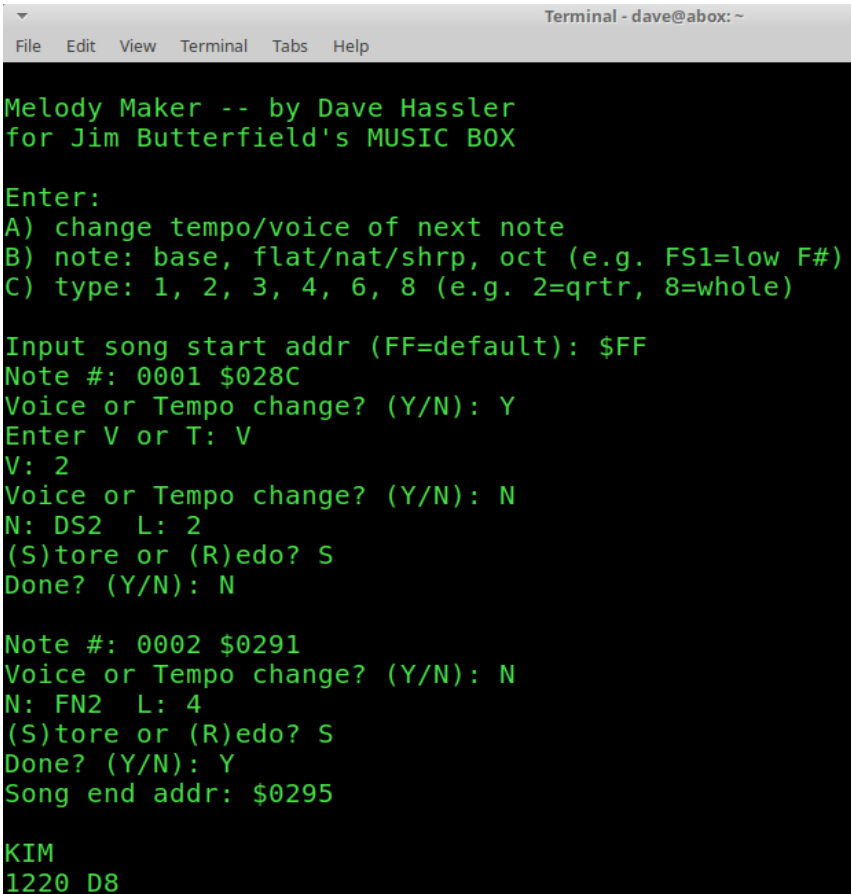
VOICE/TEMPO CHANGE

Next up is the opportunity to change the voice and tempo of your tune. The “voice” in this case being either a modified sine wave (1) or a square wave (2). Think of them as “oboe” and “80s synthesizer.”

Tempo is a different animal. The range can be from \$80 (quite slow) to \$10 (a blur). In the last section is a tempo chart that compares values with traditional tempo markings. Since you probably won't want to change these settings much after initial set-up, they are presented as a simple YES/NO choice for each note.

NOTE ENTRY

Notes are entered with four parameters: **base note value**, **modifier**, **octave**, and **length**.



```
Terminal - dave@abox: ~
File Edit View Terminal Tabs Help

Melody Maker -- by Dave Hassler
for Jim Butterfield's MUSIC BOX

Enter:
A) change tempo/voice of next note
B) note: base, flat/nat/shrp, oct (e.g. FS1=low F#)
C) type: 1, 2, 3, 4, 6, 8 (e.g. 2=qrtr, 8=whole)

Input song start addr (FF=default): $FF
Note #: 0001 $028C
Voice or Tempo change? (Y/N): Y
Enter V or T: V
V: 2
Voice or Tempo change? (Y/N): N
N: DS2 L: 2
(S)tore or (R)edo? S
Done? (Y/N): N

Note #: 0002 $0291
Voice or Tempo change? (Y/N): N
N: FN2 L: 4
(S)tore or (R)edo? S
Done? (Y/N): Y
Song end addr: $0295

KIM
1220 D8
```

Enter the first three values in a row. If you want a G right on the button of the treble clef, no flat or sharp, enter GN2. Do **NOT** hit <return> at any time! “GN2” means “I want a G natural in the second octave.” The next prompt is for the length. Enter:

- 1 – eighth note
- 2 – quarter note
- 3 – ‘dotted’ quarter note
- 4 – half note
- 6 – ‘dotted’ half note
- 8 – whole note

This should suffice for most occasions. If you need a 1/16th note, ratchet up the tempo for that note (or series of notes), give a length of 1 for the note, then reset the tempo back to it’s original value for the next note(s). Or, you could just “pretend” that a length of 1 is a 1/16, length 2 an eighth, etc. You lose your whole note that way, but sacrifices must be made in such a simple program.

I should digress... the ‘base note’ values may be from A through H. “H??” you exclaim, “There’s no ‘H’!” Here, it means “rest,” or silence. Silence/rests can have the same length and octave values as notes, *but always make the rest a natural*; e.g., HN2 4 for a half note of rest/silence.

After you enter the ‘base’ note value, the second entry tells the program if you want the note to be flat or sharp (F or S); if you want neither, press N. The third element is the octave selection. Middle C – the one exactly between the treble and bass clefs on all piano scores – is in Octave 2: CN2. The top line of the treble clef is F in the third octave, so FN3 if it’s ‘natural,’ and FS3 if it’s to be sharp (FF3 would make no sense – it’s EN3). See the chart in the appendix.

FINISH

That’s pretty much it. After the entry of each note, you’ll be asked if you are ‘done.’ That means done with entering all notes for the tune, not just done with the previous note. Enter N to keep on entering notes; enter Y to save an end-of-file marker and get an ending address *plus-1* for your song. There are two ways to save your creation, both involving text capture from your serial terminal emulator: a) make a PTP file from the KIM-1 monitor, or b) make an Apple HEX file from eWozLite. I prefer the latter method for full songs, the utility of which will become apparent in the next section.

If you want to save a PTP for game sounds, enter the least significant byte of the end address Melody Maker gave into \$17F7. Then, enter the most significant byte into \$17F8. E.g., Melody Maker gives you an end address of \$02B3. Put \$B3 into \$17F7 and \$02 into \$17F8.



Then, set the KIM-1 monitor to address \$028C (if default), or wherever you set the starting address. Start your terminal's text capture feature, then hit Q. Close/stop the text capture function, and you'll have a PTP file of your song.

To save the song in Apple HEX format, enter the eWozLite monitor at \$1220, then type the starting address, a period, then the ending address – but don't hit <return> just yet. Start your terminal's text capture feature and *then* hit <return>. When the listing finishes, close/stop the text capture, and you'll have the A1 HEX file, good for loading in later.

Part B: EDITING

Inevitably, mistakes will be made. While the Melody Maker program does not have an editing module itself, with the note chart in the Tables section it's fairly easy to make changes to the music data with eWozLite. To start examining and editing:

A) call eWozLite at \$1220
B) if the music file is not already in memory, load it with an ASCII upload from your serial terminal emulator; eWozLite is ready for input – no command is needed.

C) to display the data, type the starting address, a period, the ending address, then <return>. E.g., 28C.2F9 <ret>

D) to edit a data element, type its address, a colon, then the new value and <return>. E.g., let's say I made a typo during entry and entered a D instead of an E note. Using the picture on the last page as a reference, to change \$0292 from D7 (DN2) to CD (EN2) I would type 292:CD <ret>. You can also enter a line of bytes, each separated by a space: 292:CD FC 04 C0 ... <ret> to make changes to whatever bytes that need editing from \$0292-onward.

Let's say your song plays too slowly. "FB" is MUSIC BOX's code for tempo; again referencing the picture above, you can simply type 28F:28 <ret> to speed it up a bit.

```
1220 D8 G\
28C.2B8
028C: FE 00 FB 20
0290: FC 04 FD 01 B2 FC 03 AA 2A FC 06 C0 B9 B2 B0 AA
02A0: A5 FC 08 B9 FC 04 B2 FC 03 AD 30 FC 06 AA A5 A1
02B0: A1 A5 A5 FC 06 AA FA FF FF
DD0R
0DD0: D8

Melody Maker -- by Dave Hassler
for Jim Butterfield's MUSIC BOX

Enter A) change tempo or voice of next note (Y/N)
      B) note: base, flat/nat/shrp, oct (e.g. FS1=low F#)
      C) type: 1, 2, 3, 4, 6, 8 (e.g. 2=qrtr, 8=whole)

Input song start addr (FF=default): $02B6
Note #: 0001
Voice or Tempo change? (Y/N): _
```

APPENDING

If you have to walk away from a note entry session for a bit, save either a PTP or AHEX of the song, note the end address, and when you come back to the computer, all you need to do is:

A) load the song data
B) restart Melody Maker
C) for a starting address, use the previous end address *minus 1*. That location should hold \$FA. That code is MUSIC BOX's end-of-file marker, and there's only one in each song data file.
D) to save the appended data, save from the original start address to the new end address.

RELOCATING

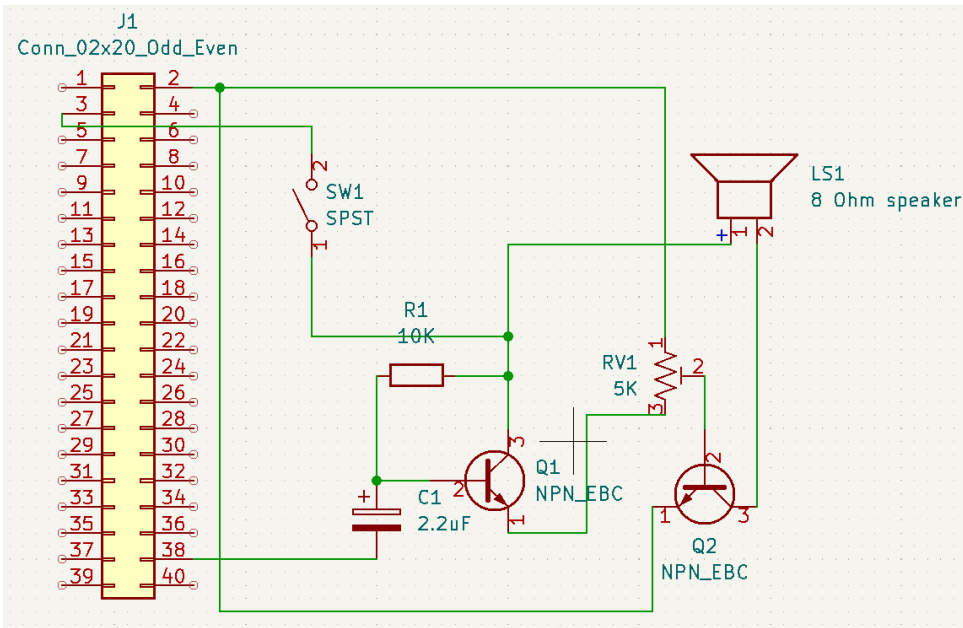
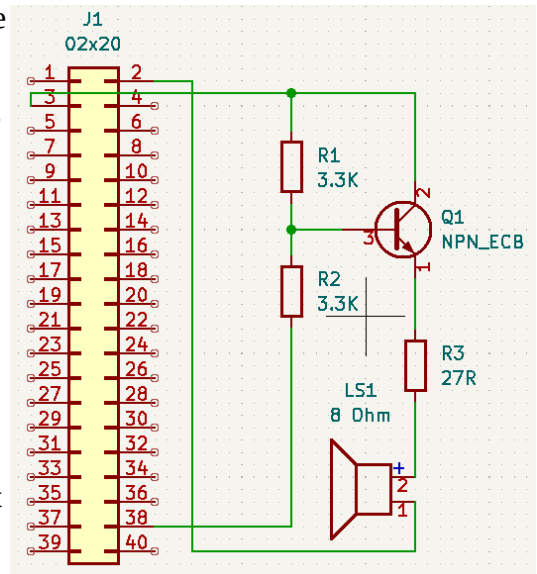
This is simplicity itself with eWozLite, since the song file is just raw data, not code. If a song is already saved as AHEX (if not, load the PTP, then examine it and save it with eWozLite first, then...), load it into your favorite text editor. Change the starting address to your new starting address (four digits), and then *take out all the other addresses*, leaving only the colon to start each line (see the example in Chapter V). Save it. eWozLite will load the result just fine, and if you want a PTP of it, just save a text capture in the usual way after loading the AHEX.

III. SOUND OUTPUT

To get audible sound, the basic idea is: feed an amplifier with PB7. It could be as simple as running wires from PB7 and GND on the expansion connector of the PAL-1 to the auxiliary input of your stereo system. This also allows you to use that system's volume control, as MUSIC BOX does not have one. Butterfield essentially says to do just that in "The First Book of KIM" listing of MUSIC BOX.

If you want to be further "authentic," you could use the little one-transistor amp described in the "KIM-1 User Manual" from MOS. To the right is the schematic, which you could breadboard or build "dead-bug style" in a few minutes. The resistor values aren't critical. 2.2Ks will substitute for the 3.3Ks, a 100 Ohm is fine in the 27 Ohm spot. It's not very loud, but gives enough volume to hear the sound OK.

If you want on-board volume control, below you'll find a basic, two-transistor audio amp. There is a little slide switch on the Vcc line to turn off the circuit, which will be desirable if you want to save a program to cassette. As for the trimmer, use a logarithmic (audio) pot for best results; a linear will work, but it's not optimal. Finally, this one can get loud!



One "better" modification is to add a pair of 10uF electrolytic capacitors around the trimmer to isolate it. This can prevent distortion, but it's not really necessary for this circuit.

Yet, if you wish to do it, for this modification connect the + side of one cap to pin 1 of Q1 and the - side to pin 3 of the trimmer; then connect the - side of the other cap to pin 2 of the trimmer and the + side to pin 2 of Q2.

For those who bought or built a second RIOT board for the PAL-1 and want to use PA0 as an output, in Jim Butterfield's original "The First Book of KIM" article he notes that the following changes should be made to the original MUSIC BOX program:

LOCATION	OLD	NEW
020D	\$43	\$01
024C	\$A7	\$FF
0255	\$27	\$00
0270	\$42	\$00

IV. APPLICATIONS and EXAMPLES

While MUSIC BOX is a fun player for basic electronic music, it also has good utility as a subroutine that a program can call to generate musical sound effects. On the unexpanded PAL-1, the high-level languages with which you can use MUSIC BOX include Tiny BASIC, VTL-2, and potentially Tiny PILOT. And good old 6502 machine code is always available, too, and works great.

If you have RAM expansion, you can use both KIM-1 BASIC (KB9) and original Apple BASIC (A1B) ported for KIM-1. PEEKing, POKEing, and CALLing to the right locations are all you need to do. Unfortunately, PAL PILOT has no way to call a machine language routine.

In any case, the first thing to do is to slightly alter MUSIC BOX (for convenience, I've provided several versions of MB, referenced below, in the archive that contains this documentation). Assuming you haven't relocated it (which is easy and must be done for the high-level languages), at location \$0219, replace the BRK (\$00) with RTS (\$60). That's it. In the already-fixed \$0115 version, it's location \$012E, etc.

Then, enter and save tunes to where you want to put them (RIOT RAM, RAM protected from the language, etc.), and call them as needed. Let's take a look at the three high-level languages that can make use of MUSIC BOX on the unexpanded PAL-1 (or microKIM, if you have one handy), see KB9 and Apple BASIC examples, and finish up the chapter with a quick look at ML integration.

Tiny BASIC

The Tiny BASIC most used by PAL-1 enthusiasts is Tom Pittman's 6502 Tiny BASIC. For this, you need to use the version of MUSIC BOX that loads at \$0115. This version has an RTS at the spot where MUSIC BOX has a BRK, and it sits right after Tiny's break-check routine located from \$0100 to \$0114 and still leaves room for more than 45 return addresses on the stack. The following data is used in all of the examples in this chapter (except machine language):

```
1780: FE 00 FC 02 AA 2A 2A FC 04 9F FA FE FF FC 02 FD
1790: 02 C8 FC 04 E8 FA
```

This sound data has two tunes in it: \$1780 and \$178B. In *exactly* this order, A) load the sound data above, B) load Tiny BASIC at \$0200, C) load the \$0115 version of MUSIC BOX. Cold start Tiny at \$0200 and then type in or load the following BASIC program via ASCII upload:

```
5 REM MUSIC BOX AT $115, DATA AT $1780
10 O=536
20 A=USR(O,415,128)
30 A=USR(O,416,23)
40 PR"WINNER!"
50 A=USR(277)
60 I=1
70 I=I+1
80 IF I<100 GOTO 70
90 A=USR(O,415,139)
100 PR"LOSER."
110 A=USR(277)
120 I=1
130 I=I+1
140 IF I<100 GOTO 130
150 PR"HA! YOU ACTUALLY *DID* WIN!"
```



```

160 A=USR(0,415,128)
170 A=USR(277)
175 GOSUB 200
180 PR"GO AGAIN (1=Y) "
182 INPUT X
184 IF X<>1 GOTO 190
186 GOTO 20
190 PR"BYE!"
195 GOTO 990
199 REM RESET I/O
200 A=USR(0,5952,254)
210 A=USR(0,5953,0)
220 A=USR(0,5954,71)
230 A=USR(0,5955,63)
240 RETURN
990 END

```

Line 10 is the decimal address of Tiny's "POKE" routine, and lines 20 and 30 deposit a 128 (\$80) into the least significant byte address of where MUSIC BOX looks for a tune to play, and a 23 (\$17) into the most significant byte location. Line 50 calls MUSIC BOX and plays the first tune. The second tune gets pointed to in line 90 and the first tune again in line 160. Line 175 points to a subroutine that resets the I/O on both A and B ports; this is necessary for user input (although printing to TTY is still OK) and also so TB won't crash on program exit.

To protect a page of RAM from Tiny BASIC (Page 13 in this case), load Tiny, cold start it at \$0200, break out and change locations \$0023 and \$0027 from \$14 to \$13. Then, load in your music data at \$1300 and, finally, warm start Tiny BASIC at \$0203, leaving exactly 2K for your program.

VTL-2 (Cu version, for unexpanded PAL-1)

With a little machine language routine, VTL-2 can play as many tunes as you have room for in RAM. The routine resides in the middle of VTL-2's full Page 2 input buffer, because you're not going to input a 128-character line anyway (72-character limit). The X register is passed to the routine from VTL-2. Here's the program:

```

* = $280
    CLD
    STA $17E5      ; temp store A,Y
    STY $17E6
    TXA
    ASL            ; multiply by 2
    TAX            ; move passed A to X
    LDA TABLE,X  ; get the LSB
    STA $018F      ; store it in Music Box
    INX
    LDA TABLE,X  ; get the MSB
    STA $0190      ; store it in Music Box
    JSR $0100      ; call Music Box
    LDA $17E5      ; reload orig. A,Y
    LDY $17E6
    RTS

```



```

RESET
    LDA #$FE          ; reset all I/O
    STA $1740
    LDA #$00
    STA $1741
    LDA #$47
    STA $1742
    LDA #$3F
    STA $1743
    RTS

```

```

TABLE
.BYTE $00,$00,$80,$17,$8B,$17

0280: D8 8D E5 17 8C E6 17 8A
0288: 0A AA BD B6 02 8D 8F 01
0290: E8 BD B6 02 8D 90 01 20
0298: 00 01 AD E5 17 AC E6 17
02A0: 60 A9 FE 8D 40 17 A9 00
02A8: 8D 41 17 A9 47 8D 42 17
02B0: A9 3F 8D 43 17 60 00 00
02B8: 80 17 8B 17

```

Since VTL-2 does not have a POKE equivalent, this allows the VTL-2 program to pass a song number, get the address of it and put it into VTL-2's version of MUSIC BOX, which loads at \$0100 and has the zero page locations changes to avoid conflicts with VTL-2. VTL-2Cu user RAM starts at \$0300. While the example data sits in RIOT RAM, if you tacked it on after the routine starting at \$02C0, you'd have 64 bytes for four sounds/tunes, which may be enough for a small game. You might also be able to lower the top of user RAM from \$0FAF to \$0F00 to gain 175 more bytes for sounds.

If you *do* lower the top of user RAM, A) load and run VTL-2 *first*, B) lower the "himem" there with *=3839 (\$0EFF – 3,071 bytes left after the change), C) break out of it, D) load the "poke-n-call" routine above, change the jump table locations, and then load the song data at \$0F00. E) come back to VTL-2 with a warm start (at \$0FC1), and F) finally load your VTL-2 program. Whew! The Tiny BASIC program above looks like this in VTL-2:

```

5 ) ML SELECT TUNE AND CALL MUSIC BOX VIA $0280
7 ?=""
10 ?="WINNER!"
20 "=640
30 >=1
40 I=1
50 I=I+1
60 #=I<500*50
62 "=673
65 >=0
70 ?="LOSER."
80 "=640
90 >=2
100 I=1

```

```

11Ø I=I+1
12Ø #=I<5ØØ*11Ø
122 "=673
125 >=Ø
13Ø ?="HA! YOU ARE REALLY A WINNER!"
14Ø "=64Ø
15Ø >=1
152 "=673
155 >=Ø
16Ø ?="GO AGAIN (Y/N) ?";
17Ø A=$
18Ø #=A=89*7
185 ?=""
19Ø ?="BYE!"

```

Lines 20 and 30 set the JSR address for our get-index-and-call routine (\$0280), will pass 1 to the X register (first tune), and jumps out. The call to location 673 (\$02A1) at lines 62-65, 122-125, and 152-155 resets the I/O ports, which is necessary to be able to take user input or print anything; it also prevents VTL-2 from crashing upon program exit. Lines 40-60 are a delay loop, 80 and 90 choose tune 2, and 140-150 reselect tune 1. You may want more delay – VTL-2 is *fast*!

Tiny PILOT

The following section applies to the Nick Vrtis program Tiny PILOT augmented by Bob Applegate and (much later) myself for use on the KIM-1 and PAL-1. This program is available on my website. Frankly, I don't think it's worth it, but if *you* do, here's what you can do to get *two* sounds:

A) raise the start of user RAM in TP to \$0700 by changing location \$0490 in Tiny to \$07

B) because MUSIC BOX uses Indirect Y addressing – which requires a location in Zero Page, and Tiny PILOT uses up almost *all* of Zero Page – you'd have to alter MB's work area so that only WORK+4 accesses Zero Page in the *one* location you have to use, like:

```

WORKZ    = $89
WORK      = $1ØØ
LIMIT     = $1Ø6
VAL2      = $1Ø9
VAL1      = $1ØA
TIMER     = $1ØB
XSAV      = $1ØC

```

then change the two references in the \$0115 version of MB of LDA (WORK+4), Y to LDA (WORKZ), Y. This is *very* similar to the headaches I'd get desperately seeking Zero Page RAM to "steal" in the VIC-20, back in the day (usually just eight whole bytes!)

C) write two tiny assembly language programs that would store the proper addresses of your tunes into MB and then call MB. Something like:

```

CLD
STA $17E5 ; temp store A,Y
STY $17E6
LDA #$A3 ; load LSB of addr of tune to play
STA $Ø19F ; store it in Music Box

```

```

LDA #$17 ; load the MSB
STA $01A0 ; store it in Music Box
JSR $0115 ; call Music Box
LDA $17E5 ; reload orig. A,Y
LDY $17E6
RTS

```

where each routine would have different addresses for the STA commands. RIOT RAM is fine for two short sounds, beeps, or buzzes, but you'd need Page 6 for anything more than 65 bytes total.

D) the third ML routine will be the RESET bit from the VTL-2 version, used the same way, and put within the Tiny PILOT script (i.e., L:C).

E) enter the locations of your two sound-playing routines and the reset routine in \$A0-A5 of Zero Page after Tiny PILOT is loaded; these are the special (and separate from "regular") A-C machine language variables' jump locations.

Tiny PILOT was not designed to call a machine language program – it's *much* more of a "dialog engine" than a "number cruncher" – although Bob Applegate added the L: statement (think "leap") to do just that. In order for TP to handle more than one string, I added a statement that cut back on the number of routines that could be accessed by Bob's routine from 26 to 3; if you want to strip out the X: statement (string rotation, at \$1780) and re-edit the L: statement to get back 26 possible calls, then feel free. Like I said: Not. A. Number. Cruncher. But, one *heck* of a story machine.

KIM-1 BASIC (KB9)

Similar to the implementation on Tiny BASIC. This works OK for me:

```

10 POKE 8256,21 : REM LSB OF MUSIC BOX
20 POKE 8257,1 : REM MSB ($0115)
30 PRINT"WINNER!"
35 POKE 415,128: POKE416,23: REM START ADDR OF 1ST TUNE
40 X=USR(0) : REM THIS CALLS MB
50 FOR I=1TO200:NEXT
60 PRINT"LOSER."
70 POKE 415,139: REM START ADDR (LSB) OF 2ND TUNE
80 X=USR(0)
90 FOR I=1TO200:NEXT
100 PRINT"WAIT! YOU ACTUALLY *DID* WIN!"
110 POKE 415,128: REM START ADDR OF 1ST TUNE
120 X=USR(0)
130 FOR I=1TO200:NEXT
140 GOSUB 200
150 INPUT "PLAY AGAIN (Y/N)";A$
160 IF LEFT$(A$,1)<>"Y" THEN END
170 GOTO 30
200 POKE 5952,254: REM SUBROUTINE TO RESET ALL I/O PORTS
210 POKE 5953,0
220 POKE 5954,71
230 POKE 5955,63
240 RETURN

```

Apple BASIC

To use MUSIC BOX with Apple BASIC, you'll need the special \$0280 version that moves MB's Zero Page use from \$E0-on to \$20-on to avoid clashes with A1B and with the WozMon. Before loading MUSIC BOX, you *must* set LOMEM to at least 801; if you set LOMEM=1024, you'll have 242 bytes for songs/noises and still have a full 4K in "lower RAM" for programs. See the discussion about loading VTL-2 if/when you move LOMEM in A1B. Of course, there's no reason you're stuck with 4K if you have RAM expansion, but:

A) always load A1B first, B) set LOMEM=xxxx , C) break out, D) load the special version of MUSIC BOX at \$0280, E) warm start A1B at \$82B3 (or \$A2B3...it's the "x2B3" that counts – "*et tu, B3?!?*") and F) load your Apple BASIC program.

In any case, to use sounds, POKE in the appropriate values, as in the KIM-1 BASIC sample program, and away you go. Here's the A1B version of the sample program:

```
3 DIM A$(10)
5 REM  MB LOADED AT 640
8 REM  POKE 778 & 779 FOR SONG LOC.
9 REM  SONG DATA AT $1780
10 PRINT "WINNER!"
15 POKE 778,128: POKE 779,23
20 CALL 640
30 FOR I=1 TO 500: NEXT I
40 PRINT "LOSER."
50 POKE 778,139
60 CALL 640
65 FOR I=1 TO 500: NEXT I
70 PRINT "WAIT! YOU *ARE* A WINNER!"
75 POKE 778,128
77 CALL 640
80 FOR I=1 TO 500: NEXT I
90 PRINT "'Y' TO GO AGAIN"
92 GOSUB 200
95 INPUT A$
100 IF A$="Y" THEN GOTO 10
110 END
200 POKE 5952,254: REM SUB TO RESET ALL I/O PORTS
210 POKE 5953,0
220 POKE 5954,71
230 POKE 5955,63
240 RETURN
```

MACHINE LANGUAGE

MUSIC BOX is so easy to use with machine language that it'd be a shame not to. :^) All you have to do is check your existing programs for Zero Page conflicts with MB (\$E0-\$EC), tack it onto the end of your program (or wherever), STA the address of the tune to play into MB, call it as a subroutine (put \$60 at location \$0219), then reset the I/O lines within your own program for input (it doesn't work well when they are reset within MUSIC BOX). Here's a little example:

```

; SONG DATA AT $028C
* = $3000
CLD
JSR $1E2F ; CALL CRLF
LDA #$57 ; W
JSR $1EA0 ; CALL OUTCH
LDA #$8C ; START OF W TUNE
STA $028A
JSR $0200 ; CALL MUSIC BOX
JSR DLAY
JSR $1E2F
LDA #$4C ; L
JSR $1EA0
LDA #$97 ; START OF L TUNE
STA $028A
JSR $0200 ; CALL MB
JSR RSET
JSR $1E2F
JSR $1F9D ; CALL GETBYT
STA $12
JSR $1E2F
LDA $12 ; PRINT ASCII VAL
JSR $1EA0 ; OF INPUT GETBYT
BRK

DLAY LDA #$00
STA $10
STA $11
LOOP CLC
LDA $10
ADC #$01
STA $10
LDA $11
ADC #$00
STA $11
CMP #$FF
BNE LOOP
RTS

RSET LDA #$FE
STA $1740
LDA #$00
STA $1741
LDA #$47
STA $1742
LDA #$3F
STA $1743
RTS

```

The BRK instruction of the main program will automatically reset the I/O ports, so no need to call RSET before exiting; alternatively, you could call START in the KIM-1 ROM. Especially if you tack MUSIC BOX onto the very end of your program, you can simply put the songs/tunes/noises after that. Just remember that you get more space for your tunes the lower on a page they can start.

Additionally, one can get the biggest “bang for the buck” by using a little ML “helper” program to play a long standalone song that MUSIC BOX on its own couldn’t handle. Let’s say you have a song with three verses, a chorus, and a bridge set up like this: V V C V C B C C Rather than input the whole song sequentially and run out of room, you only have to enter the verse, chorus, and bridge once each in Melody Maker – say at \$1300 (V), \$1360 (C) and \$13A0 (B), then call them as subroutines (almost exactly the same method as with VTL-2):

```

CLD
LDA #$13
STA $028B
JSR VERSE
JSR VERSE
JSR CHORUS
JSR VERSE
JSR CHORUS
JSR BRIDGE
JSR CHORUS
JSR CHORUS
BRK

VERSE LDA #$00
STA $028A
JSR $0200
RTS

CHORUS LDA #$60
STA $028A
JSR $0200
RTS

BRIDGE LDA #$A0
STA $028A
JSR $0200
RTS

```

V. TABLES

NOTE TABLE

NOTE	1/8 SHORT	1/4+ LONG	OCTAVE
A.....	1B	9B	4
G#.....	1D	9D	3
G.....	1F	9F	
F#.....	21	A1	
F.....	23	A3	(TOP LINE TREBLE CLEF)
E.....	25	A5	
D#.....	27	A7	
D.....	2A	AA	
C#.....	2D	AD	
C.....	30	B0	
B.....	32	B2	
A#.....	36	B6	
A.....	39	B9	
G#.....	3C	BC	2/1
G.....	40	C0	
F#.....	44	C4	
F.....	48	C8	
E.....	4D	CD	
D#.....	52	D2	
D.....	57	D7	
C#.....	5C	DC	
C.....	62	E2	(MIDDLE C, OCTAVE 2)
B.....	68	E8	
A#.....	6E	EE	
A.....	75	F5	(TOP LINE BASS CLEF OCTAVE 2)
REST.....	00	80	

The image displays musical notation for four octaves of notes in 4/2 time. The notation is organized into two systems. The first system shows Octave 1 (bass clef) and Octave 2 (bass clef). The second system shows Octave 3 (treble clef) and Octave 4 (treble clef). Notes are color-coded: red for Octave 1, green for Octave 2, blue for Octave 3, and purple for Octave 4. The notes are A, B, C, D, E, F, G, and A (Octave 4).

TEMPO CHART

The following chart is measured against quarter notes:

<u>VALUE FOR FB</u>	<u>BPM</u>	<u>TEMPO</u>
80	46	Largo
70	53	Lento
60	62	Adagio
50	74	Andante
40	92	Maestoso
38	106	Moderato
30	123	Allegretto
28	148	Allegro
22	174	Vivace
20	184	Presto
19	236	Prestissimo
18	248	"
16	272	"

SONGS

Greensleeves (listing shows multiple entries and several edits; loads fine):

```
0300: FE 00 FB 38
      : FC 02 B9 FC 04 B0 FC 02 AA FC 03 A5 23 FC 02 A5
      : FC 04 AA FC 02 B2 FC 03 C0 39 FC 02 B2 FC 04 B0
      : FC 02 B9 FC 03 B9 3C FC 02 B9 FC 04 B2 FC 02 BC
      : FC 04 CD FC 02 B9 FC 04 B0 FC 02 AA FC 03 A5 23
      : FC 02 A5 FC 04 AA FC 02 B2 FC 03 C0 39 FC 02 B2
      : FC 03 B0 32 FC 02 B9
      : FC 03 BC 44 FC 02 BC FC 06 B9 B9
      : FC 06 9F FC 03 9F 21 FC 02 A5 FC 04
      : AA FC 02 B2 FC 03 C0 39 FC 02 B2 FC 04 B0 FC 02
      : B9 FC 03 B9 3C FC 02 B9 FC 04 B2 FC 02 BC FC 06
      : CD 9F FC 03 9F 21 FC 02 A5 FC 04 AA FC 02 B2 FC
      : 03 C0 39 FC 02 B2 FC 03 B0 32 FC 02 B9 FC 03 BC
      : 44 FC 02 BC FC 06 B9 B9 FA
```

O Canada (in honor of Jim Butterfield):

```
0300: FE 00 FB 34 FC 04 B2 FC 03 AA 2A FC 06 C0 FC 02
0310: B9 B2 B0 AA A5 FC 08 B9 FC 04 B2 FC 03 AD 2D FC
0320: 06 AA FC 02 A5 A1 A1 A5 A5 FC 06 AA 39 32 FC 03
0330: B0 32 FC 02 B9 32 30 FC 03 AA 30 FC 02 B2 30 2A
0340: A5 AA B0 B2 FC 06 B9 39 32 FC 03 B0 32 FC 02 B9
0350: 32 30 FC 03 AA 30 FC 02 B2 B2 B9 AA 2A 2D 32 2D
0360: FC 08 AA FC 04 B2 FC 03 AA 2A FC 06 C0 FC 02 80
0370: FC 04 B0 FC 03 A5 25 FC 06 B9 00 00 FC 04 AA FC
0380: 03 A7 27 FC 02 A5 B0 B2 B9 FC 04 C0 B9 FC 08 B2
0390: FC 04 AA FC 03 9F 1F FC 02 A5 B0 B2 B9 FC 04 AA
03A0: C4 FC 08 C0 FA
```