

Teach KIM to Sing

Peter H Myers

Consider playing musical compositions on a KIM-1 MOS Technology microcomputer module. Although music is one of the most highly developed languages of man, it is possible within a few hours for your KIM-1 system to be educated to speak this language and to create computerized musical sounds. Furthermore, the total cost for enhancing your system to express music is minimal. It is likely that you have everything you need already.

Music essentially consists of a waveform expressed repetitively at some frequency and persisting for a particular time duration. The acoustical characteristics of an instrument greatly affect the ear's perception of the musical composition being played upon it. The basic waveform is modified by the instrument by means of phase shifting, and the production of various simple and complex harmonics which may vary greatly in amplitude.

Now consider a square wave form, an expression of only two amplitudes. The 18th century French mathematician, Joseph Fourier, discovered that the square wave is actually comprised of odd numbered harmonics of the fundamental frequency plus the fundamental frequency itself (all expressed as sine waves).

This is the same thing as saying that a square wave form is harmonically rich. This is fortunate from the point of view of this application, because it allows us to use a very simple program to have a microprocessor system generate just such a harmonically rich fundamental wave form to express any musical note. All that must be known is the note's pitch (or frequency) and duration.

Program Description

A program using this technique, written for the KIM-1 system (which uses the 6502 processor), is shown in listing 1. The duration of a note and its pitch are stored alternately in memory. This is all the main program needs to play a given piece of music. The duration, when read by the main program from memory, is always stored in the X register. The pitch is stored in the Y register.

When the program begins, the location of the first note's duration is in hexadecimal memory byte 0300. The next note's duration value is always the next *even*-numbered memory location (in the latter example, the next note's duration would be at 0302). The values for pitch are stored in the *odd*-numbered memory locations, beginning at hexadecimal byte 0301 (the next note's pitch value at 0303, and so on).

The main program starts by initializing the nonmaskable interrupt (NMI) and interrupt request (IRQ) vectors and the input and output ports, clearing the interrupt mask bit, and starting the interval timer. The note duration interrupt system is initialized (more about this later). After this preparation, the program proceeds into the note loop.

The note loop starts by reading the pitch value from memory and loading it into the Y register. The procedure then subtracts 1 from the Y register. After comparing for detection of a zero in the Y register, the loop jumps back on itself to subtract a 1 again. Thus, every seven machine cycles a 1 is subtracted from the Y register until it becomes equal to zero.

When the Y register is zero, the test

0200	A9	00	LDA	;SAVE MACHINE	023C	4C	23	02	JMP	;RESTART NOTE.
0202	8D	FA	17	STA	;ROUTINE LOC	023F	EE	20	02	INC
0205	A9	1C	LDA	;STORED IN NMI	0242	EE	20	02	INC	;NEXT NOTE
0207	8D	FB	17	STA	;VECTOR.	0245	EE	29	02	INC
020A	A9	FF	LDA	;PADD BITS	0248	EE	29	02	INC	;POINTERS.
020C	8D	01	17	STA	;INST TO "WRITE".	024B	EA		NOP	;SPACER.
020F	A9	7F	LDA	;PBDD BIT 7	024C	A9	65		LDA	;CHANGE
0211	8D	03	17	STA	;INST INP IRQ.	024E	8D	FE	17	STA
0214	A9	5F	LDA	;INTERRUPT	0251	A9	02		LDA	;TO NEW
0216	8D	FE	17	STA	;VECTOR, IRQ,	0253	8D	FF	17	STA
0219	A9	02	LDA	;SERVICE ROUTINE	0256	A9	FF		LDA	;START TIMER
021B	8D	FF	17	STA	;LOC.	0258	8D	0E	17	STA
021E	58			CLI	;ALLOW IRQ.	025B	58			CLI
021F	AD	00	03	LDA	;LOAD DURATION	025C	4C	5B	02	JMP
0222	AA			TAX	;IN X REGISTER.	025F	8D	07	17	STA
0223	A9	FF	LDA	;START TIMER	0262	4C	38	02	JMP	;SERVICE ROUTINE LOC.
0225	8D	0E	17	STA	;T / 64 / 256.	0265	8D	07	17	STA
0228	AD	01	03	LDA	;LOAD PITCH	0268	4C	14	02	JMP
022B	A8			TAY	;IN Y REGISTER.	026B	A9	00		LDA
022C	88			DEY	;NOTE TIMER	026D	8D	20	02	STA
022D	F0	03		BEQ	;LOOP 7	0270	A9	03		LDA
022F	4C	2C	02	JMP	;MACHINE CYCLES.	0272	8D	21	02	STA
0232	EE	00	17	INC	;TOGGLE OUTPUT.	0275	A9	01		LDA
0235	4C	28	02	JMP	;RESTORE LOOP.	0277	8D	29	02	STA
0238	CA			DEX	;COUNTDOWN	027A	A9	03		LDA
0239	F0	04		BEQ	;DURATION.	027C	8D	2A	02	STA
023B	58			CLI	;ALLOW IRQ.	027F	4C	00	02	JMP
										;MUSIC.

Listing 1. Music playing program for the MOS Technology KIM-1 system, which uses the 6502 processor. The program assumes that data for a particular musical composition has been stored in memory beginning at hexadecimal location 0300.

causes the program to jump out of the loop. Output port pin PA 0 is toggled to change the square wave amplitude of the tone. The program then jumps back to the beginning of the note loop, reloads the pitch value, and the process is repeated. This uninterrupted process produces a square waveform tone which emerges at PA 0. The pitch value can be calculated for a given frequency by the formula in table 1.

The initialization of the note duration interrupt system proceeded as follows. The duration value is read from memory and loaded into the X register. Also, the interval timer is started and asked to interrupt the main program after 16,384 machine cycles.

While the note loop is generating notes, the interval timer is counting down. When the counter reaches 0, the timer has timed out. This interrupts the main program by loading

the program counter with the IRQ vector. The IRQ vector is initialized to direct the processor to the interrupt service routine.

The interrupt service routine shuts off the interrupt request flag (IRQ) and subtracts 1 from the duration value in the X register. The X register is compared for detection of a zero. If X is not equal to 0, the program resets the interrupt mask bit so that interrupts can occur again. Then the routine returns to main program execution, starting the interval timer and falling into the note loop.

Each interrupt is serviced in the same way until the X register becomes equal to 0. When this occurs, that interrupt will be recognized as the end of the note. Instead of jumping back to the main program after resetting the mask bit, the interrupt service routine program continues.

The routine proceeds to increment both the X register loading pointer and the Y register loading pointer by 2. This automatically gets the next pitch and duration value pair when the main program is run again.

A pause must be inserted at the end of each note to distinguish the duration. This is accomplished by changing the IRQ vector to a different location after an interrupt request.

After this change, the interval timer is started and the program falls into a "do nothing" loop. It will stay in this wait state until an interrupt directs it to a new program location. A special routine beginning at this location will service the interrupt by turning off the IRQ request and jumping back to the beginning of the main program. The main program will play the next note for the duration requested.

The main program can play a musical composition which is 128 notes in length or less, with the pitch and duration values stored in one relative memory page. Each note will be played in the same manner previously described. The program can be revised to read note specifications to the end of memory, thus giving the capability of playing compositions longer than 128 notes. This was not done in the original program because I wanted the program to loop back and play the composition over and over.

Preparing the Music

A musical score is converted for playing by the computer in a note-by-note process. It is begun by reading each note in a melody, observing the octave it is to be played in, and then finding the corresponding frequency in a table, as may be found in the appendix. The frequency is converted into a one byte pitch value by using the formula in table 1 and then calculating the hexadecimal equivalent.

The duration of the note is calculated as a function of time in seconds by the formula in table 2. Construct a table containing this information for each note in the melodic line. Using monitor commands, store the hexadecimal values for the duration and pitch in sequential memory locations, starting at hexadecimal location 0300. The specifications for the second note will start at 0302, the third at 0304, and so on.

Whenever an interval of silence is desired, whether at the end of the piece, or embedded within, store a pitch value of 01 hexadecimal. The duration value becomes the desired length of the silent period. The pitch value of 01 produces a note at a frequency of slightly over 24 kHz, which is perceived by the human ear as silence.

$$P_n = \frac{1}{0.000014 (f_n)^{-2}}$$

f_n = frequency in Hz of note
 P_n = pitch value of note

Table 1. Formula for calculating the pitch value for a desired note. The result should be rounded to the nearest whole number, and then must be converted from decimal notation to hexadecimal for entry into the computer. The pitch value is the first byte of the note specification pair.

$$D_n = \frac{t_n}{0.016384}$$

t_n = time in seconds of note
 D_n = duration value of note

Table 2. Formula for calculating the time duration value of a note. The result must be rounded and converted to hexadecimal. The duration value is the second of the note specification byte pair.

Note	Frequency (Hz)	Pitch Value	
		Decimal	Hexadecimal
G	392.7	180	B4
G# A♭	415.7	170	AA
A	441.4	160	A0
A# B♭	467.4	151	97
B	490.0	144	90
C	522.1	135	87
C# D♭	554.5	127	7F
D	586.3	120	78
D# E♭	622.1	113	71
E	662.3	106	6A
F	701.4	100	64
F# G♭	737.7	95	5F
G	786.3	89	59

Table 3. One octave table of prepared pitch values, which begin at G above middle C. The frequencies given for each note are not quite standard, but are close enough for noncritical applications. Also, the frequencies will vary within the clock tolerances for different processors. The user may transpose these pitches down one octave by changing the output port used from PA 0 to PA 1.

	Time (seconds)	Duration value (hexadecimal)	Table 4. Set of suggested time duration values for preparing melodic data from standard musical notation. These yield a tempo with about 120 quarter notes per minute.
Whole note	2.18	85	
Half note	1.09	43	
Quarter note	0.545	20	
Eighth note	0.2725	10	
Sixteenth note	0.1362	07	

A short hexadecimal conversion table for one octave beginning at G above middle C appears in table 3. Table 4 contains a suggested set of duration values. The duration values given are appropriate for a tempo of about 120 quarter notes per minute, which usually works out to a moderately fast march tempo.

```

0300 10 87 10 87 20 87 20 B4
0308 10 6A 10 6A 20 6A 20 87
0310 10 87 10 6A 20 59 20 59
0318 10 64 10 6A 43 78 10 78
0320 10 6A 20 64 20 64 10 6A
0328 10 78 20 6A 20 87 10 87
0330 10 6A 20 78 20 B4 10 8F
0338 10 78 43 87 10 87 10 87
0340 20 87 20 B4 10 6A 10 6A
0348 20 6A 20 87 10 87 10 6A
0350 20 59 20 59 10 64 10 6A
0358 43 78 10 78 10 6A 20 64
0360 20 64 10 6A 10 78 20 6A
0368 20 87 10 87 10 6A 20 78
0370 20 B4 10 8F 10 78 43 87
0378 FF 01 FF 01 01 01 01 01

```

Listing 2. Melodic data for use by the music playing program. This was prepared from a musical score by the procedure given in the text. The tune is the American traditional song, My Darling Clementine. The pairs of digits represent the contents of one byte of memory. The user should place this data in programmable memory by monitor commands. The first byte specifies the time duration of the first note; the second byte specifies the pitch of the first note. The third byte gives the duration of the second note; the fourth byte, the second pitch, and so on.

The lowest pitch which may be obtained using the output port configuration described is the C sharp just above middle C. To obtain lower pitches, change the output port used from PA 0 to PA 1. This causes all notes to be transposed down one octave.

To prepare the hardware for music making, you should connect a jumper wire from the interval timer interrupt output to the interrupt request (IRQ) pin. Normal output is taken from PA 0 and referenced to ground. The interval timer is found at connector A pin 15, and IRQ at connector E pin 4. The output PA 0 comes from connector A pin 14. PA 1 may be found at connector A pin 4.

The audio signal present at the output port may be monitored using a number of methods. You can listen to it directly with high impedance earphones. One of the best ways is to connect the output into the auxiliary input jack of a tape recorder and use the tape monitor audio amplifier for output.

Also, since the output voltage is 1.4 V peak-to-peak, it is quite easy to use a stereo amplifier's high level input jack and play it through a stereo system.

Editor's note: When playing the output of the KIM-1 system through a high fidelity music system, the user should take care not to overtax the speaker systems by playing at high volume. The complex waveforms generated by the computer contain much more energy at high frequencies (above 10,000 Hz) than is contained by conventional music. As a consequence, the tweeter sections of speaker systems may burn out if they are made to reproduce these waveforms at high volumes. . . . RSS

It is interesting to note that the harmonics of the tones are so rich that an FM radio, when held close by the KIM-1 board, will play the tones as audio because it receives them as radio frequencies. This is, however, not the ideal system for monitoring the tones, mainly because non-musical harmonics often become louder than the desired musical notes.

For a demonstration piece I have selected an old time favorite tune. The note specifications of the melody are given as listing 2, and should be programmed into memory as listed.

Provided you have made the hardware changes needed, have a working audio monitor, and have entered listings 1 and 2 into memory properly, you are ready for the KIM to play its first musical piece. Start the execution of the program at hexadecimal memory location 026B. Reset the system by depressing the RS key, and then depressing GO.

The program will repeat the tune indefinitely, but may be stopped by depressing the ST key. Always start the program at location 026B. This will initialize the X and Y register pointers for starting at the beginning of the composition. After this subroutine has finished, control is transferred to the main program.

So it can be done. Music is being played on your KIM-1 system with a minimum of hardware. This system is now expressing a high level language that can be interpreted by all.

May you enjoy many hours of music lessons with KIM. ■