A Sampling of Techniques for Computer Performance of Music

Hal Chamberlin
29 Mead St
Manchester NH 03104

Computer music is probably one of the most talked about serious applications for home computers. By serious I mean an application that has a degree of complexity and open-endedness which can totally preoccupy experimenters and funded institutions for years. Computer performance of music is a discipline so vast that the final, "best" technique for its implementation or even a good definition of such a technique may never be discovered.

At the same time, computer music is an easy field to break into. With only minimal effort and expenditure a very impressive (to the uninitiated) music performance demonstration may be put together. With a little more work a system may be assembled which is of great value to other family members, particularly children just starting to learn music theory. Such a system could, for example, eliminate manual dexterity as a factor in a child's musical development. Finally, on the highest level, it is no longer very difficult to break into truly original research in serious performance of music by computer. The advances in digital and linear integrated circuits have made putting together the hardware system for supporting such research largely a matter of clever system design rather than brute financial strength. Programming, tempered with musical knowledge, is the real key to obtaining significant results. Thus, in the future, hobbyists working with their own systems will be making important contributions toward advancement of the computer music art.

While the scope of one article cannot fully cover such an extensive topic, it should serve to acquaint the reader with the more popular techniques, their implementation, strengths, weaknesses, and ultimate potential.

Generally, all computer music performance techniques can be classified into two generic groups. The first includes schemes in which the computer generates the sound directly. The second covers systems where the computer acts as a controller for external sound generation apparatus such as an electronic organ or sound synthesizer.

Early Techniques

Just as soon as standard commercial computers such as the IBM 709 and, later, the 1401 made their appearance, programmers started to do frivolous things with them after hours, such as playing games and music. Since elementary monotonic (one note at a time) music is just a series of tones with different frequencies and durations, and since a computer can be a very precise timing device, it did not take long for these early tinkerers to figure out how to get the machine to play such music. The fundamental concept used was that of a *timed loop.*

A timed loop is a series of machine language instructions which are carefully chosen for their execution time as well as function, and which are organized into a loop. Some of the instructions implement a counter that controls the number of passes through the loop before exiting.

Let's examine some fundamental

timed loop relationships. If the sum total execution time of the instructions in the loop is M microseconds then we have a loop *frequency* of

$$\left(\frac{10^6}{M}\right) \text{ Hertz (cycles per second).}$$

If the initial value of the decrementing counter that controls the number of loop passes is N, then the total execution time before exit from the loop is (MxN) microseconds. Thus what we really have is a "tone" with a *frequency* of

$$\left(\frac{10^6}{M}\right) \text{Hertz}$$

and a *duration* of

$$\frac{MxN}{10^6} \text{ seconds.}$$

Using different loops with more or fewer instructions will give us different Ms and thus different notes. Using different Ns when entering these loops gives different durations for the notes, and so we have satisfied the definition of elementary monotonic music.

Of course at this point the computer is merely humming to itself. Several techniques, some of them quite strange, have evolved to make the humming audible to mortals.

One such method that doesn't even require a connection to the computer is to use an AM portable radio tuned to a quiet spot on the broadcast band and held close to the computer. Viola! *[Sic]* The humming rings forth in loud, relatively clear notes. As a matter of fact, music programs using this form of output were very popular in the "early days" when most small system computers had only 256 bytes of memory and no IO peripherals except the front panel.

What is actually happening is that the internal logic circuitry with its fast rise time pulses is spewing harmonics that extend up into the broadcast band region of the radio spectrum. Since some logic gates will undoubtedly switch only once per loop iteration, the harmonics of the switching will be separated in frequency by the switching or loop frequency. Those high frequency harmonics that fall within the passband of the radio are treated as a "carrier" and a bunch of equally spaced nearly equal amplitude sidebands. The radio's detector generates an output frequency equal to the common differences of all these sidebands, which is the loop frequency and its harmonics. The timbre of the resulting tones is altered somewhat by the choice of instructions in the loop, but basically has a flat audio spectrum like that of a narrow pulse waveform. Noise and distortion arise from other logic circuitry in the computer which switches erratically with respect to the timed loops. One practical difficulty with this method is there is no clearly identifiable way to get the computer to "shut up" for rests or space between identical notes.

The Hammer-Klavier

Other early methods used some kind of output peripheral to make sound. In a demonstration of an IBM 1401 over a decade ago this was literally true: the computer played a line printer! It seems that the hookup between a 1401 central processing unit and the 1403 printer was such that software had control of the printer hammer timing. Each time a hammer was fired a pulse of sound was emitted upon impact with the paper. Using a timed loop program with a print hammer fire instruction imbedded in the loop gave a raspy but accurately pitched buzz. *[It also tended to cause IBM customer engineers great trepidation . . .*CH*]* This same scheme should also be possible on some of the small, completely software controlled dot matrix printers that are now coming on the market.

A sane approach, however, is to connect a speaker to an output port bit through an amplifier. Instructions would then be placed inside the timed loops to toggle the bit and thus produce a clean, noise-free rectangular wave.

Timed Loop Example

Let's look at an example of a timed loop music playing program, not so much for its musical value (which is negligable), but for some insight into what is involved, and also to introduce some terms. The MOS Technology 6502 microprocessor will be used for these examples. These programs are designed to run on a KIM-1 system, and should run on most other 6502-based systems with very minor modifications. Motorola 6800 users should be able to easily convert the programs into 6800 machine language. 8080 users will benefit most because successful conversion indicates a thorough understanding of the concepts involved.

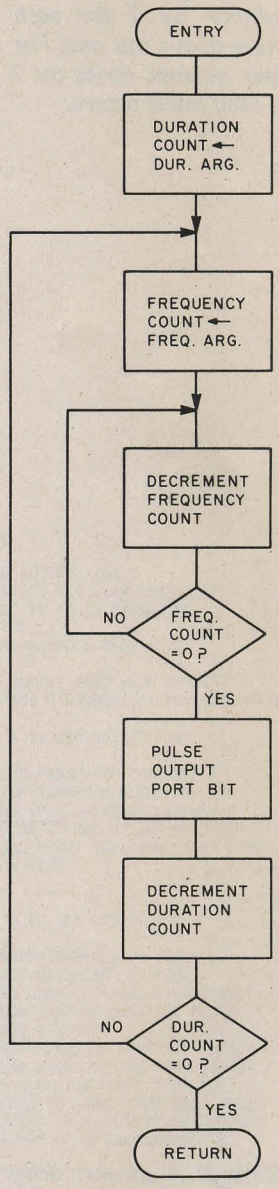


Figure 1: A basic tone generation subroutine. There are two nested loops in this routine: the first, or inner loop controls the frequency (or pitch) of the note to be generated, while the second, outer loop controls the duration of the note. A train of square waves is generated at the output port bit which is used to drive the circuit in figure 2 to produce an audible tone.

Note	Frequency (Hz)
Middle C	261.62
C#	277.18
D	293.66
D#	311.13
E	329.63
F	349.23
F#	369.99
G	391.99
G#	415.30
A	440.00
A#	466.16
B	493.88

Table 1: Equally tempered scale note frequencies in Hertz. In order to determine frequencies of notes in the higher octaves, multiply by 2 for each octave above this one. For lower octaves, divide by 2 for each lower octave.

The heart of the program is the tone generation subroutine which will be named TONE. Ideally, such a routine would accept as input two arguments: one related to the pitch of the note and the other controlling the duration. With such a subroutine available, playing a piece of music amounts to simply fetching the arguments from a "song" table in memory and calling the routine for each note to be played.

As mentioned previously, we could have a separate, carefully timed loop for each different tone frequency needed. TONE would then call the proper one based on the pitch parameter. Indeed this approach is very accurate (to within 1 μs on the 6502) but a great deal of memory is consumed for the 30 or so notes typically required. It also lacks flexibility. (This will be discussed later.) A better approach is to embed a second, *waiting* loop to control the execution time of one pass through the outer loop, and thus the tone's frequency. Figure 1 is a flowchart illustrating this. When using this scheme, the frequency argument directly determines the number of times through the inner, waiting loop and the duration parameter directly determines the number of times through the outer, tone generation loop.

Now, how are the argument values determined to get the frequencies and durations desired? First the execution time of the nested loops must be determined. In the KIM-1 with a 1 MHz clock and a 6502 the tightest inner waiting loop that can be written is 5 μs, assuming that the inner loop count (frequency argument) is 256 or less and that it is held in a register. The total time spent in the loop is [(5xM)-1]) microseconds, where M is the frequency argument and the -1 is due to the shorter execution time of an unsuccessful branch. (The observant reader will note that the execution time of some 6502 instructions is altered if they cross a memory "page boundary"; thus, an assumption of no page crossing is made.) But there is still the time required for a pass through the outer loop to output a pulse and decrement the duration counter. This is termed "loop overhead." For an example, let's say that the loop overhead is 25 μs. As a result, the total outer loop time is [(5xM)-1+25], or [(5xM)+24] microseconds which is the period of the audio waveform output. In order to determine the M required for a particular note, a table of note frequencies (see table 1) is consulted. Then the equation,

$$M=\frac{\left(\frac{10^6}{F}-24\right)}{5}$$

where F is the desired frequency, is solved for the nearest integer value of M. Lower frequency notes are preferred so that the percentage error incurred due to rounding M is minimized. The duration argument is actually a count of the number of audio tone cycles which are to be generated for the note, and thus its value is dependent on the tone frequency as well as the duration. Its value can be determined from the relation N=DxF, where N is the duration argument, D is the duration in seconds, and F is the note frequency in Hertz.

As a complete example, let's assume that an eighth note G# an octave above middle C is to be played, and that the piece is in 4/4 time with a metronome marking of 80 beats per minute. Since an eighth note in this case is one half of a beat, the duration will be

$$\frac{0.5x60}{80}$$

or 0.375 seconds. The note table shows that the frequency of G# an octave above middle C is 830.6 Hz, which yields a frequency argument of 236. The duration argument is 311. So if TONE is called with these parameters, a nice G# eighth note will be produced.

Now let's go a step further and look at a practical "music peripheral" and TONE subroutine. Figure 2 shows a circuit for driving a speaker from any kind of TTL compatible

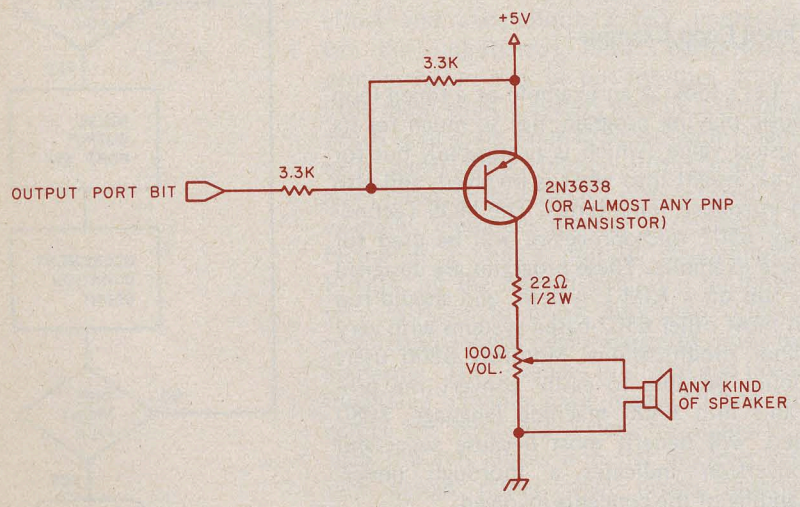


Figure 2: A speaker driver circuit designed to accept square or rectangular waves and produce audible tones through a loudspeaker. In this particular application the circuit is driven from an output port bit of a KIM-1 microcomputer, although the circuit can accept any TTL compatible output port bit. When the input to the circuit is a logical 0 level, the transistor turns on and drives the speaker. When the input is a logical 1, the transistor turns off and current to the speaker is interrupted.

Wave Duty Cycle	Harmonics									
	Fund	2	3	4	5	6	7	8	9	10
1/2	1.00	0	0.333	0	0.200	0	0.143	0	0.111	0
1/3	1.00	0.500	0	0.250	0.200	0	0.143	0.125	0	0.100
1/4	1.00	0.707	0.333	0	0.162	0.236	0.143.	0	0.111	0.141
1/5	1.00	0.841	0.561	0.259	0	0.173	0.240	0.210	0.116	0
1/6	1.00	0.867	0.667	0.433	0.200	0	0.143	0.217	0.222	0.173

Table 2: Harmonic amplitudes of rectangular waves. Note that, unlike square waves, asymmetrical rectangular waves contain even numbered harmonics. This simple technique of varying the duty cycle of such waves can have an appreciable effect on the timbre of the resulting sound.

output port bit, including those found in the 6530 "combo chips" used in the KIM-1. When the output port bit is a logic 0 level, the transistor turns on and drives a current determined by the volume control setting through the speaker. When the bit is a logic 1, the current is interrupted. Larger speakers or even a high fidelity speaker system will give a richer timbre to the lower pitched tones. The AUX input to a sound system may also be used instead of the transistor circuit. Using a patch cord, connect the shield to the common terminal of the power supply and the center conductor to the output port bit through a 10 K to 100 K isolation resistor.

Listing 1 shows an assembled listing of a practical timed loop tone generation subroutine for the 6502 microprocessor. Several refinements beyond the flowcharted example have been made to improve tone quality and flexibility. The inner waiting loop has been split into two loops. The first loop determines the length of time that the output rectangular waveform is to be a logic 1 and the second loop determines the 0 time. If both loops receive the same frequency argument (which they do as written) and the loop time of both loops is the same, then a symmetrical square wave output is produced. However, if one or more "do nothing" instructions is inserted into one of the two loops, the output waveform will become nonsymmetrical. The significance of this is that the rectangular waveform's duty cycle affects its harmonic spectrum, and thus its timbre. In particular, there is a large audible difference between a 50%-50% duty cycle (square wave) and a 25%-75% duty cycle. Table 2 lists the harmonic structure of some possible rectangular waves. As a result, some control over the timbre can be exercised if a separate TONE subroutine is written for each "voice" desired. Unfortunately, if this is done the frequency arguments will have to be recomputed since the outer loop time will then be altered.

Real music also possesses *dynamics*, which are the changes in overall volume during a performance. Furthermore, the *amplitude envelope* of a tone is an important contributor to its overall subjective timbre. The latter term refers to rapid changes in volume during a single note. This is the case with a piano note, which builds up rapidly at the beginning and slowly trails off thereafter. Of course the setup described thus far has no control over either of these parameters: the volume level is constant, and the envelope of each note is rectangular with sudden onset and termination.

```
                ;       TONE SUBROUTINE FOR 6502
                ;       ENTER WITH FREQUENCY PARAMETER IN ACCUMULATOR
                ;       DURATION PARAMETER STORED AT LOCATION DUR (LOW PART) AND
                ;       DUR+1 (HIGH PART) WHICH IS ASSUMED TO BE IN PAGE ZERO
                ;       ROUTINE USES A, X, AND DESTROYS DUR
                ;       LOOP TIME = 10*(FREQ PARAMETER)+44 MICROSECONDS

1700            MPORT   =       X'1700      ; ADDRESS OF OUTPUT PORT WITH SPEAKER
00E0            DUR     =       X'E0        ; ARBITRARY PAGE 0 ADDRESS OF DURATION PARM

0100 A2FF       TONE:   LDX     #X'FF       ; SEND ALL 1'S TO THE OUTPUT PORT
0102 8E0017             STX     MPORT
0105 AA                 TAX                 ; TRANSFER FREQ PARAMETER TO INDEX X
0106 CA         WHIGH:  DEX                 ; WAIT LOOP FOR WAVEFORM HIGH TIME
0107 D0FD               BNE     WHIGH       ; TIME IN THIS LOOP = 5*FREQ PARAMETER
0109 F000               BEQ     .+2         ; WAIT 15 STATES TO MATCH TIME USED TO
010B F000               BEQ     .+2         ; DECREMENT AND CHECK DURATION COUNT AFTER
010D F000               BEQ     .+2         ; WAVEFORM LOW TIME
010F F000               BEQ     .+2
0111 F000               BEQ     .+2
0113 A200               LDX     #0          ; SEND ALL 0'S TO THE OUTPUT PORT
0115 8E0017             STX     MPORT
0118 AA                 TAX                 ; TRANSFER FREQ PARAMETER TO INDEX X
0119 CA         WLOW:   DEX                 ; WAIT LOOP FOR WAVEFORM LOW TIME
011A D0FD               BNE     WLOW        ; TIME IN THIS LOOP = 5*FREQ PARAMETER
011C C6E0               DEC     DUR         ; DECREMENT LOW PART OF DURATION COUNT
011E D005               BNE     TIMWAS      ; BRANCH IF NOT RUN OUT
0120 C6E1               DEC     DUR+1       ; DECREMENT HIGH PART OF DURATION COUNT
0122 D0DC               BNE     TONE        ; GO DO ANOTHER CYCLE OF THE TONE IF NOT 0
0124 60                 RTS                 ; RETURN WHEN DURATION COUNT RUNS OUT
0125 F000       TIMWAS: BEQ     .+2         ; WASTE 7 CYCLES TO EQUAL TIME THAT WOULD
0127 F000               BEQ     .+2         ; HAVE BEEN SPENT IF HIGH PART OF DUR WAS
0129 D0D5               BNE     TONE        ; DECREMENTED AND GO DO ANOTHER CYCLE
```

Listing 1: An assembled listing of a practical timed loop tone generation subroutine for the 6502 microprocessor. This routine is an elaboration of the flowchart shown in figure 1 which allows the user to generate nonsymmetrical rectangular waves. Experimenting with the wave's duty cycle affects the harmonic content of the resulting tone and creates many interesting aural effects.

By graduating to a more sophisticated music peripheral, control of dynamics and amplitude envelopes can be achieved with a timed loop music program. The secret is to use a *digital to analog* converter connected to all eight bits of the output port. A digital to analog converter (DAC) does just what its name implies: it accepts a binary number from the output port as input and generates a corresponding DC voltage as its output.

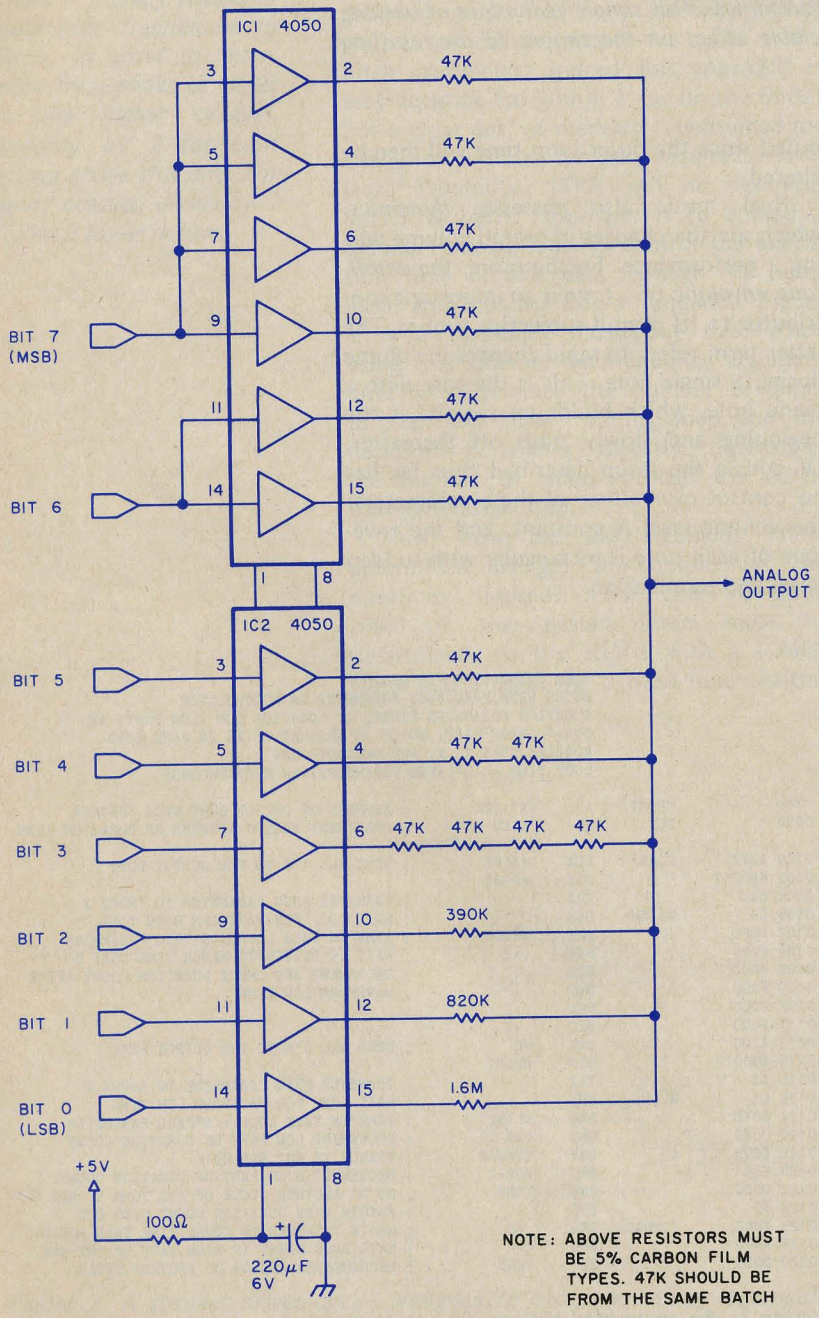


Figure 3: An 8 bit digital to analog converter (DAC). This circuit accepts an 8 bit binary number from the output port and generates a corresponding DC voltage as its output. The output voltage from this circuit is equal to ((I/255)-x5) V, where I is the decimal equivalent of the 8 bit input which can take on any value from 0 to 225.

The circuit in figure 3, which can be used with any TTL compatible output port, gives an output voltage

$$V=\left(\frac{I}{255}\right)x5$$

where I is the binary number input between 0 and 255. When working with this kind of DAC, it is convenient to regard the binary number, I, as a fraction between 0 and 1 rather than an integer. The benefit of this will become apparent later when calculations will be performed to arrive at the value of I. The output of the DAC must be used with a sound system or the amplifier circuit in figure 8, not the simple transistor speaker driver circuit in figure 2.

As written, the TONE subroutine (see listing 1) alternately sends 0 and 255 to the output port with the music peripheral. With a DAC connected to that port, voltages of 0 and 5 V will be produced for the low and high portions of the rectangular wave. If instead 0 and 127 were output, the DAC would produce only 0 and 2.5 V giving a rectangular wave with about half the amplitude. This in turn produces a less loud tone, and so control over dynamics is possible by altering the byte stored at hexadecimal 101.

Arbitrary amplitude envelopes are also made possible by continuously exercising control over the amplitude during a note. Simple envelope shapes such as a linear attack and decay can be computed in line while the note is being sounded. A more general method is to build a table in memory describing the shape. Such a table can be quickly referenced during note playing. Great care must be taken, however, to insure that loop timing is kept stable when the additional instructions necessary to implement amplitude envelopes are added.

More Complex Techniques

Even if all of the improvements mentioned above were fully implemented, the elementary timed loop approach falls far short of significant musical potential. The primary limitations are a narrow range of tone colors and restriction to monotonic performance. The latter difficulty may be alleviated through the use of a multitrack tape recorder to combine separate parts, but this requires an investment in noncomputer hardware and is certainly not automatic. Also, unpitched percussive sounds such as drum beats are generally not possible. Musicians, too, will probably notice a host of other limitations such as lack of vibrato and

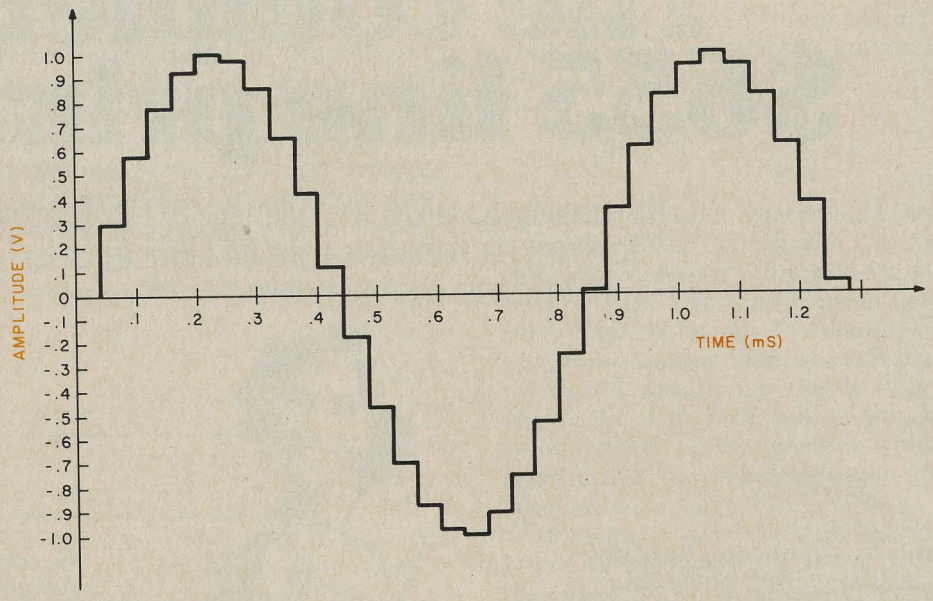


Figure 4: A sine wave as it would appear at the output from the digital to analog converter shown in figure 3. Each step in the approximation of this wave is called a sample. This particular illustration shows a 1.2 kHz sine wave sampled at a rate of 25,000 samples per second. The resulting waveform is only a very rough approximation of the original, but low pass filtering can improve accuracy (see figure 5 and text).

other subtle variations. All of these shortcomings may be overcome by allowing the computer to compute the entire sound waveform in detail at its own speed.

The one fundamental concept that makes direct waveform computation possible is the *sampling theorem*. Any waveform, no matter how simple or complex, can be reconstructed from a rapid series of discrete voltage values by means of a digital to analog converter such as the one used earlier. As an example, let's try to generate an accurate sine wave using a DAC. If this can be done, it follows from the Fourier (harmonic) theorem that any other waveform may also be synthesized.

Figure 4 shows a sine wave as it would appear at the DAC output. Each step on the approximation to the sine wave is termed a *sample*, and the frequency with which these samples emerge from the DAC is the *sample rate*. An attempt is being made in the example to generate a 1.2 kHz sine wave at a sample rate of 25 kHz, or one sample every 40 μs. Obviously this is a very poor sine wave, a fact that can be easily demonstrated with a distortion analyzer.

Before giving up, let's look at the frequency spectrum of this staircase-like wave on a spectrum analyzer. The spectral plot in figure 5 shows a strong frequency component at 1.2 kHz which is the sine wave we are trying to synthesize. Also present are the distortion component frequencies due to the sampling process. Since all of the distortion components are much higher in frequency than the desired signal, they may be easily removed with a sharp low pass filter. After filtering, the distortion analyzer will confirm that a smooth, pure sine wave is all that remains.

What will happen if the sine wave frequency is increased but the sampling frequency remains constant? With even fewer samples on each sine wave cycle the waveform from the DAC will appear even more distorted. The lowest frequency distortion product is the one of concern since it is the most difficult to filter out. Its frequency is FD=(FS-f) Hertz, where FD is the lowest distortion component frequency, FS is the sampling frequency, and f is the sine wave signal frequency. Thus as f increases, FD decreases until they merge at f=FS/2. This frequency is termed the *Nyquist frequency* and is the highest theoretical frequency that may be synthesized. Any attempt to synthesize a higher frequency will result in the desired signal being filtered out and the distortion frequency emerging instead. This situation is termed *aliasing* because the desired signal frequency has been replaced by a distortion component alias frequency. Operating close to the Nyquist frequency requires a very sharp filter to separate the signal from the distortion. With practical filters, signal frequencies up to 1/4 to 1/3 of the sampling frequency are realizable.

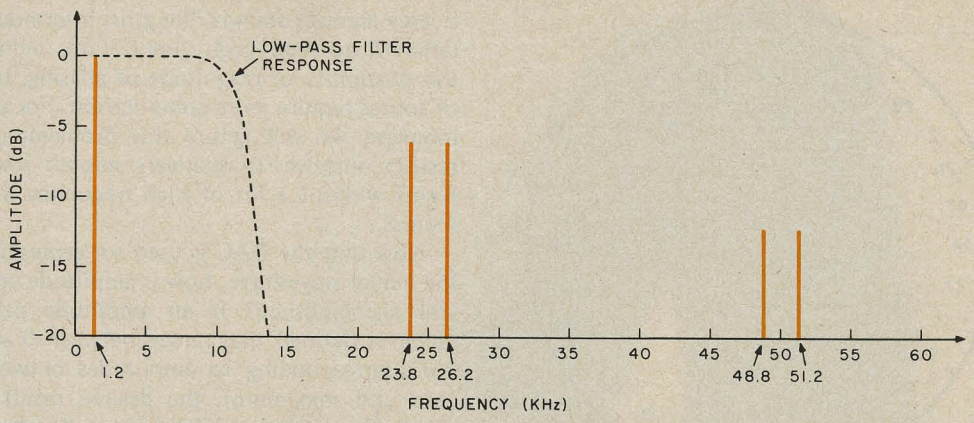


Figure 5: The spectral plot of the staircase-like sine wave approximation shown in figure 4. This frequency versus amplitude graph indicates a strong frequency component at 1.2 kHz, the frequency of the sine wave. Normally, this would be the only frequency component to appear on a plot like this, but the presence of steeply rising steps in this waveform approximation introduces distortion components at higher frequencies, as shown.

Since any sound, whether it is a pitched tone or unpitched sound, is actually a combination of sine waves, it follows that any possible sound may be produced by a DAC. The only limitation is the upper frequency response, which may be made as high as desired by increasing the sample rate. The low frequency response has no limit, and extends down to DC.

There is another form of distortion in DAC generated sounds which cannot be filtered out, since it is spread throughout the frequency spectrum. *Quantization noise* is due to the fact that a DAC cannot generate voltages that are exact samples on the desired waveform. An 8 bit converter, for example, has only 256 possible output voltage values. When a particular voltage is needed, the nearest available value will have to be used. The theoretical signal to noise ratio when using a perfect DAC is related to the number of bits by the equation S/N= (6xM)+4 decibels where M is the number of bits. A practical DAC may be as much as 6 db worse, but a cheap 8 bit unit can yield nearly 50 db, which is as good as many tape recorders. When using 12 bits or more, the DAC will outperform even the best professional recorders. Thus it is apparent that computed waveforms can, in theory, be used to generate very high quality music; so high, in fact, that conventional audio equipment is hard pressed to reproduce it.

Now that we have the tools, let's see how the limitations of computer music mentioned earlier can be overcome. For tones of definite pitch, the timbre is determined by the waveshape and the amplitude envelope. Concentrating on the waveshape, it should be apparent that a waveform table in memory repeatedly dumped into the DAC will produce an equivalent sound waveform. Each table entry becomes a sample, and the entire table represents one cycle of the waveform. The frequency of the resulting tone will be FS/N where FS is the sampling frequency (rate at which table entries are sent to the DAC) and N is the number of entries in the table. To get other frequencies, either the sample rate or the number of table entries must be changed.

There are a number of reasons why the sample rate should remain constant, so the answer is to change the effective table length. If the table dump routine were modified to skip every other entry, the result would be an effective halving of table size and thus doubling of the tone frequency. If the table is fairly long, such as 256 entries, a number of frequencies are possible by skipping an integer number of entries.

To get musically accurate frequencies, it is necessary to be able to skip a fractional number of table entries. At this point the concept of a *table increment* is helpful in dealing with programming such an operation. First, the table is visualized as a circle with the first entry conceptually following the last as in figure 6. A pointer locates a point along the circular table which represents the sample last sent to the DAC. To find what should be sent to the DAC next, the table pointer is moved clockwise a distance equal to the table increment. The frequency of the resulting tone is now

$$\frac{FS \times I}{N}$$

where FS and N are as before and I is the increment.

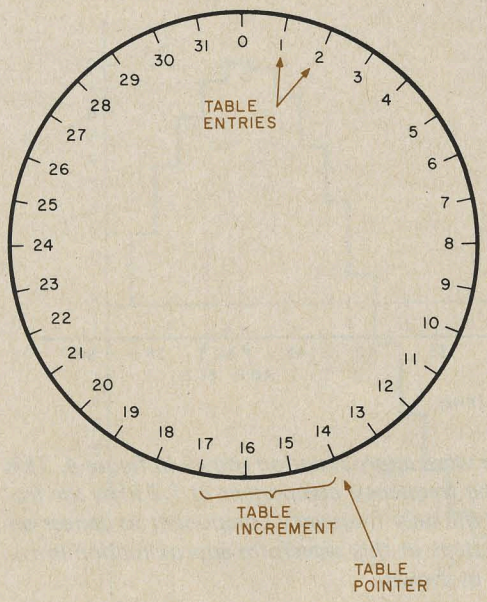


Figure 6: Diagrammatic representation of the circular table used for storing the waveform "template." The technique illustrated here is that of storing a large number of samples of one cycle of a musical waveform in memory as a table which wraps around itself in circular fashion. A pointer is used to point to the next sample to be extracted. In order to create a waveform with a given frequency, the program is designed to skip a fractional number of table entries to get the next sample value. This fractional number is called the table increment value. The process is continued around the table for one revolution to create a complete waveform. The cycle around the table is repeated until the duration counter decrements to zero.

With integer increments, the pointer always points squarely to an entry. With mixed number increments, the pointer also will take on a fractional part. The sensible thing to do is to interpolate between the table entries on either side of the pointer to arrive at an accurate value to give to the DAC. This is indeed necessary to assure high quality; but simply choosing the nearest entry may be acceptable in some cases, particularly if the table is very large.

There is one elusive pitfall in this technique. The table may contain the tabulation of any waveform desired, subject to one limitation: a nonzero harmonic component of the waveform must not exceed the Nyquist frequency, FS/2. This can easily happen with the larger table increments (higher frequency tones), the result being aliasing of the upper harmonics. Theoretically this is a severe limitation. Often a small amount of aliasing is not objectionable, but a large amount sounds like gross intermodulation distortion. High sample rates reduce the possibility or magnitude of aliasing, but of course require more computation. For the moment, we will ignore this problem and restrict ourselves to relatively smooth waveforms without a lot of high frequency harmonics.

Now that the DAC is used for generating the actual waveshape, how is amplitude control accomplished? If an amplitude parameter is defined that ranges between 0 and 1.0 (corresponding to amplitudes between zero and maximum), the desired result is obtained by simply multiplying each sample from the table by this amplitude parameter and sending the product to the DAC. Things are nice and consistent if the table entries are also considered as fractions between −1 and +1 because then the product has a range between −1 and +1 which is directly compatible with the DAC. (Note that the DAC in figure 3 is unipolar. It can be considered bipolar if +2.5 V output is the zero reference and the sign bit is inverted.)

The last major hurdle is the generation of simultaneous tones. Obviously, two simultaneous tones may be generated by going through two tables, outputting to two separate DACs, and mixing the results with an audio mixer. This is relatively simple to do if the sample rates of the two tones are the same. Actually, all the audio mixer does is to *add* the two input voltages together to produce its output, but a very important realization is that the addition can also be done in the computer before the output conversion by the DAC! The two samples are simply added together with an ADD instruction, the sum is divided by two (to constrain it to the range of −1 to +1), and the result sent to a single DAC. This holds true for any number of simultaneous tones! The only requirement is that the composite samples not overflow the −1 to +1 range that the DAC can accept. Rather than dividing the sum, it is best to adjust the amplitude factors of the individual "voices" to prevent overflow. So now we have the tools necesary to generate an ensemble of tones, each one possibly having its own waveform, amplitude envelope, and loudness relative to the others. Indeed, this is all that is necessary to simulate a typical organ.

Up to this point the timbre (waveform) of a tone has been determined by the contents of a fixed waveform table. Truly interesting musical notes change their timbre during the duration of the note. A reasonable alternative to switching between similar tables for implementing this is to build the tone from harmonic components. Each harmonic component of the tone is simply

Listing 2: A program which, in conjunction with tables 3, 4 and 5, generates four simultaneous musical voices, each with a different waveform and volume level. The program is designed for use with the 6502 processor coupled to an 8 bit unsigned digital to analog converter (DAC) like the one shown in figure 3.

```
                ;       THIS PROGRAM PLAYS MUSIC IN 4-PART HARMONY ON THE KIM-1 OR
                ;       OTHER 6502 BASED SYSTEM USING AN 8-BIT UNSIGNED
                ;       DIGITAL-TO-ANALOG CONVERTER CONNECTED TO AN OUTPUT PORT.  TUNED
                ;       FOR SYSTEMS WITH A 1 MHZ CRYSTAL CLOCK.  DOES NOT USE THE ROR
                ;       INSTRUCTION.
                ;       SONG TABLE IS AT "SONG"
                ;       ENTRY POINT IS AT "MUSIC"

0000                    .=      0               ; ORG AT PAGE 0 LOCATION 0

1700            DAC     =       X'1700          ; OUTPUT PORT ADDRESS WITH DAC
1701            DACDIR  =       X'1701          ; DATA DRIECTION REGISTER FOR DAC PORT
1780            AUXRAM  =       X'1780          ; ADDRESS OF EXTRA 128 BYTES OF RAM IN 6530
1C22            KIMMON  =       X'1C22          ; ENTRY POINT TO KIM KEYBOARD MONITOR

0000 00         V1PT:   .BYTE   0               ; VOICE 1 WAVE POINTER, FRACTIONAL PART
0001 0000               .WORD   WAV1TB          ; INTEGER PART AND WAVE TABLE BASE
0003 00         V2PT:   .BYTE   0               ; VOICE 2
0004 0000               .WORD   WAV2TB
0006 00         V3PT:   .BYTE   0               ; VOICE 3
0007 0000               .WORD   WAV3TB
0009 00         V4PT:   .BYTE   0               ; VOICE 4
000A 0000               .WORD   WAV4TB

000C 0000       V1IN:   .WORD   0               ; VOICE 1 INCREMENT (FREQUENCY PARAMETER)
000E 0000       V2IN:   .WORD   0               ; VOICE 2
0010 0000       V3IN:   .WORD   0               ; VOICE 3
0012 0000       V4IN:   .WORD   0               ; VOICE 4

0014 00         DUR:    .BYTE   0               ; DURATION COUNTER
0015 0000       NOTES:  .WORD   0               ; NOTES POINTER
0017 0002       SONGA:  .WORD   SONG            ; ADDRESS OF SONG
0019 0000       INCPT:  .WORD   0               ; POINTER FOR LOADING UP V1NT - V4NT
001B 0C00       INCA:   .WORD   V1IN            ; INITIAL VALUE OF INCPT
001D 5200       TEMPO:  .WORD   82              ; TEMPO CONTROL VALUE, TYPICAL VALUE FOR
                                                ; 3:4 TIME, 100 BEATS PER MINUTE, DUR=64
                                                ; DESIGNATES A QUARTER NOTE
0100                    .=      X'100           ; START PROGRAM CODE AT LOCATION 0100

                ;       MAIN MUSIC PLAYING PROGRAM

0100 A9FF       MUSIC:  LDA     #X'FF           ; SET PERIPHERAL A DATA DIRECTION
0102 8D0117             STA     DACDIR          ; REGISTER TO OUTPUT
0105 D8                 CLD                     ; INSURE BINARY ARITHMETIC
0106 A517               LDA     SONGA           ; INITIALIZE NOTES POINTER
0108 8515               STA     NOTES           ; TO BEGINNING OF SONG
010A A518               LDA     SONGA+1
010C 8516               STA     NOTES+1
010E A000       MUSIC1: LDY     #0              ; SET UP TO TRANSLATE 4 NOTE ID NUMBERS
0110 A51B               LDA     INCA            ; INTO FREQUENCY DETERMINING WAVEFORM TABLE
0112 8519               STA     INCPT           ; INCREMENTS AND STORE IN V1IN - V4IN
0114 B115               LDA     (NOTES),Y       ; GET DURATION FIRST
0116 F03C               BEQ     ENDSNG          ; BRANCH IF END OF SONG
0118 C901               CMP     #1              ; TEST IF END OF SONG TABLE SEGMENT
011A F029               BEQ     NXTSEG          ; BRANCH IF SO
011C 8514               STA     DUR             ; OTHERWOSE SAVE DURATION IN DUR
011E E615       MUSIC2: INC     NOTES           ; DOUBLE INCREMENT NOTES TO POINT TO THE
0120 D002               BNE     MUSIC3          ; NOTE ID OF THE FIRST VOICE
0122 E616               INC     NOTES+1
0124 B115       MUSIC3: LDA     (NOTES),Y       ; GET A NOTE ID NUMBER
0126 AA                 TAX                     ; INTO INDEX X
0127 B520               LDA     FRQTAB+1,X      ; GET LOW BYTE OF CORRESPONDING FREQUENCY
0129 9119               STA     (INCPT),Y       ; STORE INTO LOW BYTE OF VOICE INCREMENT
012B E619               INC     INCPT           ; INDEX TO HIGH BYTE
012D B51F               LDA     FRQTAB,X        ; GET HIGH BYTE OF FREQUENCY
012F 9119               STA     (INCPT),Y       ; STORE INTO HIGH BYTE OF VOICE INCREMENT
0131 E615               INC     NOTES           ; DOUBLE INCREMENT NOTES TO POINT TO THE
0133 D002               BNE     MUSIC4          ; NOTE ID OF THE NEXT VOICE
0135 E616               INC     NOTES+1
0137 E619       MUSIC4: INC     INCPT           ; INDEX TO NEXT VOICE INCREMENT
0139 A519               LDA     INCPT           ; TEST IF 4 VOICE INCREMENTS DONE
013B C914               CMP     #V4IN+2
013D D0E5               BNE     MUSIC3          ; LOOP IF NOT
013F 205701             JSR     PLAY            ; PLAY THIS GROUP OF NOTES
0142 4C0E01             JMP     MUSIC1          ; GO LOAD UP NEXT SET OF NOTES

0145 C8         NXTSEG: INY                     ; END OF SEGMENT, NEXT TWO BYTES POINT TO
0146 B115               LDA     (NOTES),Y       ; BEGINNING OF THE NEXT SEGMENT
0148 48                 PHA
0149 C8                 INY                     ; GET BOTH SEGMENT ADDRESS BYTES
014A B115               LDA     (NOTES),Y
014C 8516               STA     NOTES+1         ; THEN STORE IN NOTES POINTER
014E 68                 PLA
014F 8515               STA     NOTES
0151 4C0E01             JMP     MUSIC1          ; GO START INTERPRETING NEW SEGMENTT

0154 4C221C     ENDSNG: JMP     KIMMON          ; END OF SONG, RETURN TO MONITOR
                ;       4 VOICE PLAY SUBROUTINE
```

a sine wave with an amplitude dependent on the waveform of the resulting tone. Giving a different amplitude envelope to each harmonic is equivalent to smoothly changing the timbre during the note. The aliasing problem mentioned earlier can also be solved by simply omitting any harmonics that become too high in frequency.

Dynamic timbre variation can also be accomplished by a *digital filter* which does the same thing to a sampled waveform that a real inductance-capacitance filter does to a normal waveform. A digital filter is simply a subroutine which accepts a sample value as an argument and gives back a sample value which represents the filtered output. The equations used in the subroutine determine the filter type, and other arguments determine the cutoff frequency, Q, etc. This is a fascinating subject which deserves its own article.

What about other, unpitched sounds? They too can be handled with a few simple techniques. Most sounds in this category are based in part on random noise. In sampled form, random white noise with a uniform frequency spectrum is simply a stream of random numbers. For example, a fairly realistic snare drum sound may be generated by simply giving the proper amplitude envelope to pure white noise. Other types of drum sounds may be generated by using a digital filter to shape the frequency spectrum of the noise. A resonant type of digital filter would be used for tom-toms and similar semipitched drums, for example. A high pass filter is useful for simulating brush and cymbal sounds. An infinite number of variations are possible. This is one area where direct computation of sound waveforms really shines.

The sampling theorem works both ways also. Any waveform may be converted into digital samples with an analog to digital converter (ADC) with no loss of information. The only requirement is that the signal being sampled have no frequency components higher than half of the sampling frequency. This may be accomplished by passing the signal to be digitized through a sharp low pass filter prior to presenting it to the ADC. Once sound is in digitized form, literally anything may be done to it. A simple (in concept) application is intricate editing of the sound with a graphic display, light pen and large capacity disk. The sound may be analyzed into harmonic components and the result or a transformation of it applied to a synthesized sound. Again, this is an area that deserves its own article.

Listing 2, continued:

```
0157 A000       PLAY:   LDY   #0           ; SET Y TO ZERO FOR STRAIGHT INDIRECT
0159 A61D               LDX   TEMPO        ; SET X TO TEMPO COUNT
                                           ; COMPUTE AND OUTPUT A COMPOSITE SAMPLE

015B 18         PLAY1:  CLC                ; CLEAR CARRY
015C B101               LDA   (V1PT+1),Y   ; ADD UP 4 VOICE SAMPLES
015E 7104               ADC   (V2PT+1),Y   ; USING INDIRECT ADDRESSING THROUGH VOICE
0160 7107               ADC   (V3PT+1),Y   ; POINTERS INTO WAVEFORM TABLES
0162 710A               ADC   (V4PT+1),Y   ; STRAIGHT INDIRECT WHEN Y INDEX = 0
0164 8D0017             STA   X'1700       ; SEND SUM TO DIGITAL-TO-ANALOG CONVERTER
0167 A500               LDA   V1PT         ; ADD INCREMENTS TO POINTERS FOR
0169 650C               ADC   V1IN         ; THE 4 VOICES
016B 8500               STA   V1PT         ; FIRST FRACTIONAL PART
016D A501               LDA   V1PT+1
016F 650D               ADC   V1IN+1
0171 8501               STA   V1PT+1       ; THEN INTEGER PART
0173 A503               LDA   V2PT         ; VOICE 2
0175 650E               ADC   V2IN
0177 8503               STA   V2PT
0179 A504               LDA   V2PT+1
017B 650F               ADC   V2IN+1
017D 8504               STA   V2PT+1
017F A506               LDA   V3PT         ; VOICE 3
0181 6510               ADC   V3IN
0183 8506               STA   V3PT
0185 A507               LDA   V3PT+1
0187 6511               ADC   V3IN+1
0189 8507               STA   V3PT+1
018B A509               LDA   V4PT         ; VOICE 4
018D 6512               ADC   V4IN
018F 8509               STA   V4PT
0191 A50A               LDA   V4PT+1
0193 6513               ADC   V4IN+1
0195 850A               STA   V4PT+1
0197 CA                 DEX                ; DECREMENT & CHECK TEMPO COUNT
0198 D008               BNE   TIMWAS       ; BRANCH TO TIME WASTE IF NOT RUN OUT
019A C614               DEC   DUR          ; DECREMENT & CHECK DURATION COUNTER
019C F00C               BEQ   ENDNOT       ; JUMP OUT IF END OF NOTE
019E A61D               LDX   TEMPO        ; RESTORE TEMPO COUNT
01A0 D0B9               BNE   PLAY1        ; CONTINUE PLAYING
01A2 D000       TIMWAS: BNE   .+2          ; 3  WASTE 12 STATES
01A4 D000               BNE   .+2          ; 3
01A6 D000               BNE   .+2          ; 3
01A8 D0B1               BNE   PLAY1        ; 3  CONTINUE PLAYING
01AA 60         ENDNOT: RTS                ; RETURN
                                           ; TOTAL LOOP TIME = 114 STATES = 8770 HZ

01AB            P1END   =                  ; DEFINE BEGINNING ADDRESS FOR THIRD PART
                                           ; OF SONG TABLE
```

Sampled Waveform Example

It should be obvious by now that while these sampled waveform techniques are completely general and capable of high quality, there can be a great deal of computation required. Even the most powerful computers in existence would be hard pressed to compute samples for a significant piece of music with many voices and all subtleties implemented at a rate fast enough for direct output to a DAC and speaker. Typically the samples are computed at whatever rate the program runs and are saved on a mass storage device. After the piece has been "computed," a playback program retrieves the samples and sends them to the DAC at a uniform high rate.

Most microprocessors are fast enough to do a limited amount of sampled waveform computation in real time. The 6502 is one of the best 8 bit machines in this capacity due to its indexed and indirect addressing modes and its overall high speed. The example program shown in listing 2 has the inherent capability to generate four simultaneous voices, each with a different waveform and volume level. In order to make the whole thing fit in a basic KIM-1, however, only one waveform table is actually used.

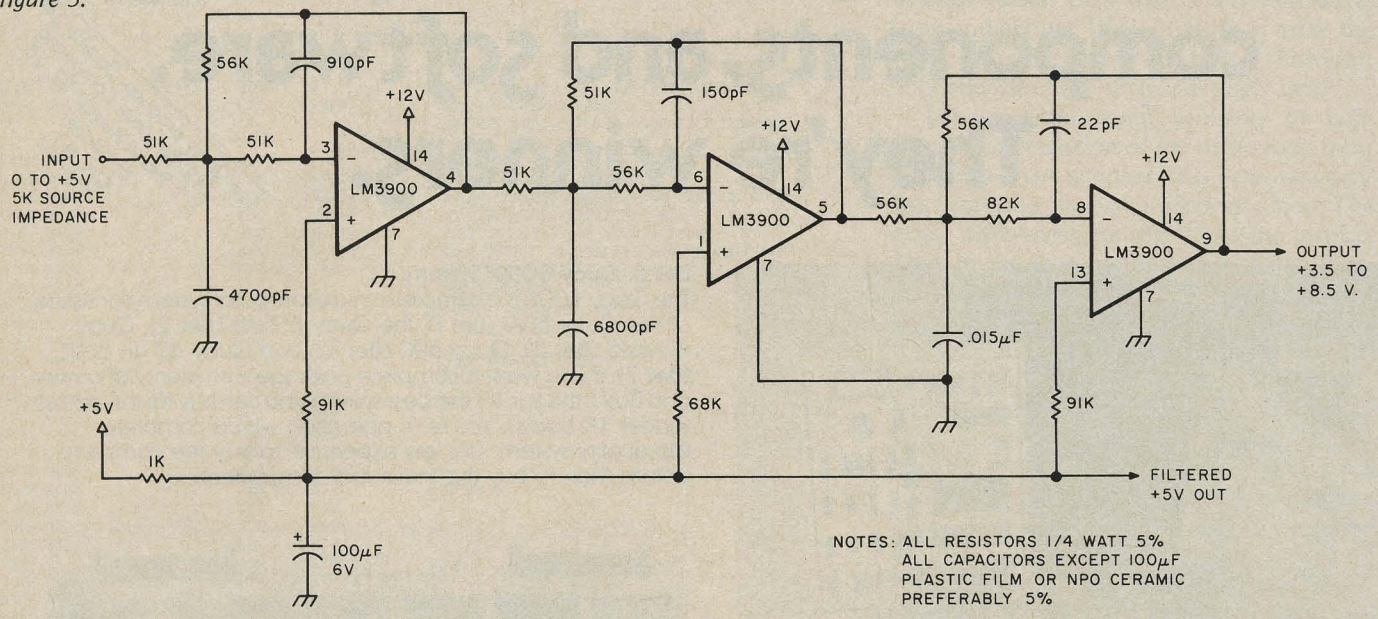
This program could probably be considered as a variation of the timed loop technique, since the sample rate is determined by the execution time of a particular loop. The major differences are that all of the instructions in the loop perform an essential function and that the loop time is constant regardless of the notes being played. Using the program as shown on a full speed (1.0 MHz) 6502 gives a sample rate of 8.77 kHz, which results in a useful upper frequency limit of 3 kHz. The low pass filter in figure 7 coupled with the DAC in figure 3 and audio system or amplifier in figure 8 are all the specialized hardware necessary to run the program with full 4 part harmony.

The program consists of two major routines: MUSIC and PLAY. MUSIC steps through the list of notes in the song table and sets up DUR and V1IN thru V4IN for the PLAY routine. PLAY simultaneously plays the four notes specified by V1IN thru V4IN for the time period specified by DUR. Another variable, TEMPO, in page zero controls the overall tempo of the music independently of the durations specified in the song table. The waveform tables for the four voices are located at WAV1TB thru WAV4TB and require 256 bytes (one memory page) each. The actual waveform samples stored in the table have already been scaled so that when four of them are added up there is no possibility of overflow.

The song table has an entry for each musical "event" in the piece. An entry requires five bytes, the first of which is a duration parameter. By suitable choice of the TEMPO parameter in page 0, "round" (in the binary sense) numbers may be used for duration parameters of common note durations. A duration parameter of 0 signals the end of the song, in which case the program returns to the monitor. A duration parameter of 1 is used to specify a break in the sequential flow of the song table. In this case the next two bytes point to the continuation of the table elsewhere in memory. This feature was necessary to deal with the fragmented memory of the KIM-1, but has other uses as well. All other possible duration values are taken literally and are followed by four bytes which identify the notes to be played by each voice. Each note ID points to a location in the note frequency table which in turn contains a 2 byte frequency parameter for that note which is placed in V1IN thru V4IN.

The PLAY routine is optimized for speed,

Figure 7: A sharp low pass filter with 3 kHz cutoff. This circuit is used to filter out the high frequency distortion illustrated in figure 5.



NOTES: ALL RESISTORS 1/4 WATT 5%
ALL CAPACITORS EXCEPT 100μF
PLASTIC FILM OR NPO CERAMIC
PREFERABLY 5%

```
                ;       NOTE FREQUENCY TABLE FOR 8.772 KHZ SAMPLE RATE
                ;       RANGE FROM C2 (65.41 HZ) TO C6 (1046.5 HZ)
                ;                                  ID NOTE  FREQ.  INCR.
001F 0000       FRQTAB: .BYTE  0,0          ;   0  SILENCE
0021 01E9               .BYTE  1,233        ;   2  C2    65.405  1.9089
0023 0206               .BYTE  2,6          ;   4  C2#   69.295  2.0224
0025 0225               .BYTE  2,37         ;   6  D2    73.415  2.1427
0027 0245               .BYTE  2,69         ;   8  D2#   77.783  2.2701
0029 0268               .BYTE  2,104        ;  10  E2    82.408  2.4051
002B 028C               .BYTE  2,140        ;  12  F2    87.308  2.5481
002D 02B3               .BYTE  2,179        ;  14  F2#   92.498  2.6996
002F 02DC               .BYTE  2,220        ;  16  G2    97.998  2.8601
0031 0308               .BYTE  3,8          ;  18  G2#  103.83   3.0302
0033 0336               .BYTE  3,54         ;  20  A2   110.00   3.2104
0035 0367               .BYTE  3,103        ;  22  A2#  116.54   3.4013
0037 039A               .BYTE  3,154        ;  24  B2   123.47   3.6035
0039 03D1               .BYTE  3,209        ;  26  C3   130.81   3.8178
003B 040B               .BYTE  4,11         ;  28  C3#  138.59   4.0448
003D 0449               .BYTE  4,73         ;  30  D3   146.83   4.2854
003F 048A               .BYTE  4,138        ;  32  D3#  155.57   4.5402
0041 04CF               .BYTE  4,207        ;  34  E3   164.82   4.8102
0043 0519               .BYTE  5,25         ;  36  F3   174.62   5.0962
0045 0566               .BYTE  5,102        ;  38  F3#  185.00   5.3992
0047 05B8               .BYTE  5,184        ;  40  G3   196.00   5.7203
0049 060F               .BYTE  6,15         ;  42  G3#  207.65   6.0604
004B 066C               .BYTE  6,108        ;  44  A3   220.00   6.4208
004D 06CD               .BYTE  6,205        ;  46  A3#  233.08   6.8026
004F 0735               .BYTE  7,53         ;  48  B3   246.94   7.2071
0051 07A3               .BYTE  7,163        ;  50  C4   261.62   7.6356
0053 0817               .BYTE  8,23         ;  52  C4#  277.18   8.0897
0055 0892               .BYTE  8,146        ;  54  D4   293.66   8.5707
0057 0915               .BYTE  9,21         ;  56  D4#  311.13   9.0804
0059 099F               .BYTE  9,159        ;  58  E4   329.63   9.6203
005B 0A31               .BYTE  10,49        ;  60  F4   349.23  10.1924
005D 0ACC               .BYTE  10,204       ;  62  F4#  369.99  10.7984
005F 0B71               .BYTE  11,113       ;  64  G4   391.99  11.4405
0061 0C1F               .BYTE  12,31        ;  66  G4#  415.30  12.1208
0063 0CD7               .BYTE  12,215       ;  68  A4   440.00  12.8416
0065 0D9B               .BYTE  13,155       ;  70  A4#  466.16  13.6052
0067 0E6A               .BYTE  14,106       ;  72  B4   493.88  14.4142
0069 0F45               .BYTE  15,69        ;  74  C5   523.24  15.2713
006B 102E               .BYTE  16,46        ;  76  C5#  554.36  16.1794
006D 1124               .BYTE  17,36        ;  78  D5   587.32  17.1414
006F 1229               .BYTE  18,41        ;  80  D5#  622.26  18.1607
0071 133E               .BYTE  19,62        ;  82  E5   659.26  19.2406
0073 1462               .BYTE  20,98        ;  84  F5   698.46  20.3847
0075 1599               .BYTE  21,153       ;  86  F5#  739.98  21.5969
0077 16E2               .BYTE  22,226       ;  88  G5   783.98  22.8811
0079 183E               .BYTE  24,62        ;  90  G5#  830.60  24.2417
007B 19AF               .BYTE  25,175       ;  92  A5   880.00  25.6831
007D 1B36               .BYTE  27,54        ;  94  A5#  932.32  27.2103
007F 1CD4               .BYTE  28,212       ;  96  B5   987.76  28.8283
0081 1E8B               .BYTE  30,139       ;  98  C6  1046.5   30.5426
0083            POEND   =      .            ; DEFINE BEGINNING ADDRESS FOR SECOND PART
                                            ; OF SONG TABLE
```

Table 3: Note frequency table used in conjunction with listing 2. This table is for a sample rate of 8.772 kHz. The range of the notes used is from 65.41 Hz (for C2) to 1046.5 Hz (for C6).

because its loop time determines the sample rate. Essentially, the routine maintains four pointers (V1PT thru V4PT) to the four waveform tables. Each pointer consists of three bytes in order of increasing significance. The first byte is the "fractional part" of the pointer, and the second byte is the integer part which is also the lower half of an address in the waveform table. The third byte is the upper address which normally remains constant. Waveform table lookup is considerably simplified by using the indirect addressing mode of the 6502 with these pointers. Note that the fractional part of the pointer is ignored when the table lookup takes place, since interpolation is much too slow for a real time routine.

During each sample, waveform table entries for each voice are fetched, added up, and sent to the digital to analog converter output port. Then the increment (VxIN) is added (double precision) to each pointer (VxPT). Wraparound from the end of a waveform table to the beginning is automatically taken care of due to the fact that the table occupies a full memory page. Finally, the tempo counter is decremented and checked. If the tempo counter is zero, it is restored and the duration counter is decremented and checked. If it is also zero the note is finished and PLAY returns. The net result is that TxD samples are computed and sent out for the event, where T is the tempo parameter and D is the duration parameter. Note that, unlike the earlier timed loop example, there is no interaction between the duration parameter and the note frequencies being played.

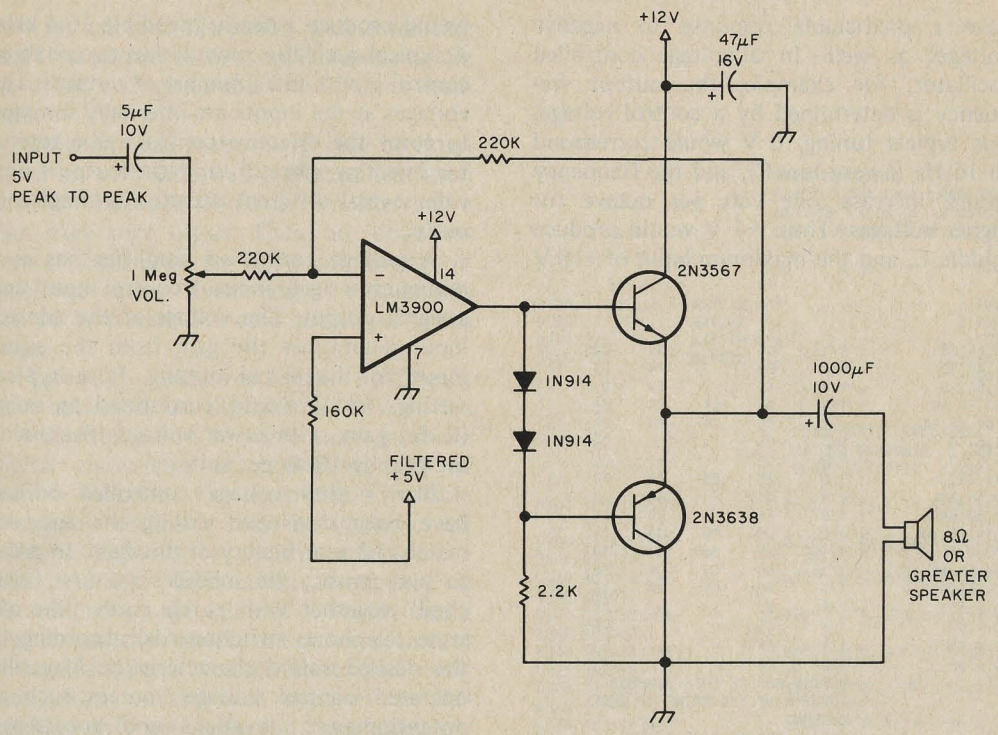


Figure 8. An inexpensive, wide band low power audio amplifier. This circuit, when coupled with the circuits in figures 3 and 7, is all the experimenter needs to create music with his or her microprocessor.

How does it sound? With the waveform table shown and a reasonably good speaker system, the result sounds very much like an electronic organ, such as a Hammond. There is a noticeable background noise level due to compromises such as prescaled waveforms and lack of interpolation in the tables, but it is not objectionable. The pitches are very accurate, but there is some beating on chords due to compromises inherent in the standard equally tempered musical scale. Also there are noticeable clicks between notes due to the time taken by the MUSIC routine to set up the next set of notes. All in all the program makes a good and certainly inexpensive basis for the "family music application" mentioned earlier.

Synthesizer Control Techniques

So far we have discussed techniques in which the computer itself generates the sound. It is also possible to interface a computer to specialized sound generation hardware and have it act as a *control* element.

The most obvious kind of equipment to control is the standard, modular, voltage controlled sound synthesizer. Since the interface characteristics of nearly all synthesizers and modules are standardized, a computer interface to such equipment could be used with nearly any synthesizer in common use.

Generally speaking, the function of a voltage controlled module is influenced by one or more DC control voltages. These are usually assumed to be in the range of 0 to +10 volts, although some modules will

```
                        ;       SONG TABLE
                        ;       EACH MUSICAL EVENT CONSISTS OF 5 BYTES
                        ;       THE FIRST IS THE DURATION OF THE EVENT IN UNITS ACCORDING TO
                        ;       THE VALUE OF "TEMPO", ZERO DENOTES THE END OF THE SONG.
                        ;       THE NEXT 4 BYTES CONTAIN THE NOTE ID OF THE 4 VOICES, 1 THROUGH
                        ;       4.  0 INDICATES SILENCE FOR THE VOICE.

0200                            .=      X'200           ; START SONG AT 0200

                        ;       SONG TABLE FOR THE STAR SPANGLED BANNER BY FRANCIS SCOTT KEY
                        ;       AND J. STAFFORD SMITH
                        ;       DURATION COUNT = 64 FOR QUARTER NOTE

0200 604A000032  SONG:  .BYTE  96,74,0,0,50         ; 3/8    C5              C4   1
0205 104400002C         .BYTE  16,68,0,0,44         ; 1/16   A4              A3
020A 4040000024         .BYTE  64,64,0,0,36         ; 1/4    G4              F3   2
020F 4044000024         .BYTE  64,68,0,0,36         ; 1/4    A4              F3
0214 404A000022         .BYTE  64,74,0,0,34         ; 1/4    C5              E3
0219 80544E441E         .BYTE  128,84,78,68,30      ; 1/2    F5   D5   A4    D3   3
021E 305C52441C         .BYTE  48,92,82,68,28       ; 3/16   A5   E5   A4    C#3
0223 105800401C         .BYTE  16,88,0,64,28        ; 1/16   G5             G4   C#3
0228 4054003C1E         .BYTE  64,84,0,60,30        ; 1/4    F5             F4   D3   4
022D 4044003C1E         .BYTE  64,68,0,60,30        ; 1/4    A4             F4   D3
0232 4048403C28         .BYTE  64,72,64,60,40       ; 1/4    B4   G4   F4   G3
0237 804A403A32         .BYTE  128,74,64,58,50      ; 1/2    C5   G4   E4   C4   5
023C 204A000032         .BYTE  32,74,0,0,50         ; 1/8    C5             C4
0241 204A000032         .BYTE  32,74,0,0,50         ; 1/8    C5             C4
0246 605C544424         .BYTE  96,92,84,68,36       ; 3/8    A5   F5   A4    F3   6
024B 2058004028         .BYTE  32,88,0,64,40        ; 1/8    G5             G4   G3
0250 4054003C2C         .BYTE  64,84,0,60,44        ; 1/4    F5             F4   A3
0255 80524A4032         .BYTE  128,82,74,64,50      ; 1/2    E5   C5   G4   C4   7
025A 304E46002E         .BYTE  48,78,70,0,46        ; 3/16   D5   B@4       B@3
025F 10524A402E         .BYTE  16,82,74,64,46       ; 1/16   E5   C5   G4   B@3
0264 40544A442C         .BYTE  64,84,74,68,44       ; 1/4    F5   C5   A4   A3   8
0269 4054000003C        .BYTE  64,84,0,0,60         ; 1/4    F5             F4
026E 404A000032         .BYTE  64,74,0,0,50         ; 1/4    C5             C4
0273 404400002C         .BYTE  64,68,0,0,44         ; 1/4    A4             A3   9
0278 403C000024         .BYTE  64,60,0,0,36         ; 1/4    F4             F3
027D 304A000032         .BYTE  48,74,0,0,50         ; 3/16   C5             C4
0282 104400002C         .BYTE  16,68,0,0,44         ; 1/16   A4             A3
0287 403C000024         .BYTE  64,60,0,0,36         ; 1/4    F4             F3   10
028C 4044000024         .BYTE  64,68,0,0,36         ; 1/4    A4             F3
0291 404A000022         .BYTE  64,74,0,0,34         ; 1/4    C5             E3
0296 80544E441E         .BYTE  128,84,78,68,30      ; 1/2    F5   D5   A4    D3   11
029B 305C52441C         .BYTE  48,92,82,68,28       ; 3/16   A5   E5   A4    C#3
```

Table 4: This song table is an encoding of "The Star Spangled Banner" in 4 part harmony which is used by the program in listing 2. Each musical event in the table consists of five bytes. The first byte represents the duration of the event in units, according to the value of the "tempo" (0 denotes the end of the song). The next four bytes contain the note identifications of the four voices (0 indicates silence for the voice).

have a predictable response to negative voltages as well. In a voltage controlled oscillator, for example, the output frequency is determined by a control voltage. For typical tuning, 0 V would correspond to 16 Hz (a very low C), and the frequency would increase one volt per octave for higher voltages. Thus, +4 V would produce middle C, and the maximum input of +10 V would produce a nearly inaudible 16.4 kHz. A typical oscillator module has two or three control inputs and a number of outputs. The voltages at the inputs are internally summed to form the effective control value (useful for injecting vibrato), and the outputs provide several different waveforms simultaneously.

Table 4, continued:

```
02A0 105800401C          .BYTE  16,88,0,64,28      ; 1/16   G5        G4   C#3
02A5 4054003C1E          .BYTE  64,84,0,60,30      ; 1/4    F5        F4   D3   12
02AA 4044003C1E          .BYTE  64,68,0,60,30      ; 1/4    A4        F4   D3
02AF 4048403C28          .BYTE  64,72,64,60,40     ; 1/4    B4   G4   F4   G3
02B4 804A403A32          .BYTE  128,74,64,58,50    ; 1/2    C5   G4   E4   C4   13
02B9 204A000032          .BYTE  32,74,0,0,50       ; 1/8    C5             C4
02BE 204A000032          .BYTE  32,74,0,0,50       ; 1/8    C5             C4
02C3 605C544424          .BYTE  96,92,84,68,36     ; 3/8    A5   F5   A4   F3   14
02C8 2058004028          .BYTE  32,88,0,64,40      ; 1/8    G5        G4   G3
02CD 2054003C2C          .BYTE  32,84,0,60,44      ; 1/8    F5        F4   A3
02D2 80524A4032          .BYTE  128,82,74,64,50    ; 1/2    E5   C5   G4   C4   15
02D7 304E46002E          .BYTE  48,78,70,0,46      ; 3/16   D5   B@4       B@3
02DC 10524A402E          .BYTE  16,82,74,64,46     ; 1/16   E5   C5   G4   B@3
02E1 40544A442C          .BYTE  64,84,74,68,44     ; 1/4    F5   C5   A4   A3   16
02E6 405400003C          .BYTE  64,84,0,0,60       ; 1/4    F5             F4
02EB 404A000032          .BYTE  64,74,0,0,50       ; 1/4    C5             C4
02F0 404400002C          .BYTE  64,68,0,0,44       ; 1/4    A4             A3   17
02F5 403C000024          .BYTE  64,60,0,0,36       ; 1/4    F4             F3
02FA 01                  .BYTE  1                  ; DEFINE END OF THIS SEGMENT
02FB 8300                .WORD  POEND              ; ADDRESS OF BEGINNING OF NEXT
                                                   ; SEGMENT
0083                     .=     POEND              ; ORG AT END OF PAGE 0 CODE
0083 305C544428          .BYTE  48,92,84,68,40     ; 3/16   A5   F5   A4   G3
0088 105C544428          .BYTE  16,92,84,68,40     ; 1/16   A5   F5   A4   G3
008D 405C544424          .BYTE  64,92,84,68,36     ; 1/4    A5   F5   A4   F3   18
0092 405E544628          .BYTE  64,94,84,70,40     ; 1/4    B@5  F5   B@4  G3
0097 4062544A2C          .BYTE  64,98,84,74,44     ; 1/4    C6   F5   C5   A3
009C 8062544A2C          .BYTE  128,98,84,74,44    ; 1/2    C6   F5   C5   A3   19
00A1 205E544628          .BYTE  32,94,84,70,40     ; 1/8    B@5  F5   B@4  G3
00A6 205C54442C          .BYTE  32,92,84,68,44     ; 1/8    A5   F5   A4   A3
00AB 4058524032          .BYTE  64,88,82,64,50     ; 1/4    G5   E5   G4   C4   20
00B0 405C54443C          .BYTE  64,92,84,68,60     ; 1/4    A5   F5   A4   F4
00B5 405E524640          .BYTE  64,94,82,70,64     ; 1/4    B@5  E5   B@4  G4
00BA 805E58461A          .BYTE  128,94,88,70,26    ; 1/2    B@5  G5   B@4  C3   21
00BF 405E52461A          .BYTE  64,94,82,70,26     ; 1/4    B@5  E5   B@4  C3
00C4 605C4A4424          .BYTE  96,92,74,68,36     ; 3/8    A5   C5   A4   F3   22
00C9 20584A4028          .BYTE  32,88,74,64,40     ; 1/8    G5   C5   G4   G3
00CE 40544A3C2C          .BYTE  64,84,74,60,44     ; 1/4    F5   C5   F4   A3
00D3 80524A4032          .BYTE  128,82,74,64,50    ; 1/2    E5   C5   G4   C4   23
00D8 204E00362E          .BYTE  32,78,0,54,46      ; 1/8    D5             D4   B@3
00DD 20524A3A2E          .BYTE  32,82,74,58,46     ; 1/8    E5   C5   E4   B@3
00E2 40544A3C2C          .BYTE  64,84,74,60,44     ; 1/4    F5   C5   F4   A3   24
00E7 40443C0036          .BYTE  64,68,60,0,54      ; 1/4    A4   F4        D4
00EC 01                  .BYTE  1                  ; DEFINE END OF THIS SEGMENT
00ED AB01                .WORD  P1END              ; ADDRESS OF BEGINNING OF NEXT
                                                   ; SEGMENT
01AB                     .=     P1END              ; ORG AT END OF PAGE 1 CODE
01AB 4048403C28          .BYTE  64,72,64,60,40     ; 1/4    B4   G4   F4   G3
01B0 804A403A1A          .BYTE  128,74,64,58,26    ; 1/2    C5   G4   E4   C3   25
01B5 404A000032          .BYTE  64,74,0,0,50       ; 1/4    C5             C4
01BA 40544A4424          .BYTE  64,84,74,68,36     ; 1/4    F5   C5   A4   F3   26
01BF 4054464028          .BYTE  64,84,70,64,40     ; 1/4    F5   B@4  G4   G3
01C4 20544A442C          .BYTE  32,84,74,68,44     ; 1/8    F5   C5   A4   A3
01C9 20524A442C          .BYTE  32,82,74,68,44     ; 1/8    E5   "    "    "
01CE 404E463C2E          .BYTE  64,78,70,60,46     ; 1/4    D5   B@4  F4   B@3  27
01D3 404E463C2E          .BYTE  64,78,70,60,46     ; 1/4    D5   B@4  F4   B@3
01D8 404E4A3E2C          .BYTE  64,78,74,62,44     ; 1/4    D5   C5   F#4  A3
01DD 4058464028          .BYTE  64,88,70,64,40     ; 1/4    G5   B@4  G4   G3   28
01E2 205E460028          .BYTE  32,94,70,0,40      ; 1/8    B@5  B@4       G3
01E7 205C44002C          .BYTE  32,92,68,0,44      ; 1/8    A5   A4        A3
01EC 205840002E          .BYTE  32,88,64,0,46      ; 1/8    G5   G4        B@3
01F1 01                  .BYTE  1                  ; DEFINE END OF THIS SEGMENT
01F2 8017                .WORD  AUXRAM             ; ADDRESS OF BEGINNING OF NEXT
                                                   ; SEGMENT (IN 6530 RAM)
1780                     .=     AUXRAM             ; ORG AT BEGINNING OF 6530 RAM
1780 20543C0030          .BYTE  32,84,60,0,48      ; 1/8    F5   F4        B3
1785 40544A4432          .BYTE  64,84,74,68,50     ; 1/4    F5   C5   A4   C4   29
178A 40524A401A          .BYTE  64,82,74,64,26     ; 1/4    E5   C5   G4   C3
178F 204A000032          .BYTE  32,74,0,0,50       ; 1/8    C5             C4
1794 204A00002E          .BYTE  32,74,0,0,46       ; 1/8    C5             B@3
1799 60544A442C          .BYTE  96,84,74,68,44     ; 3/8    F5   C5   A4   A3   30
179E 2058004032          .BYTE  32,88,0,64,50      ; 1/8    G5        G4   C4
17A3 205C004440          .BYTE  32,92,0,68,64      ; 1/8    A5        A4   G4
17A8 205E004640          .BYTE  32,94,0,70,64      ; 1/8    B@5       B@4  G4
17AD 80625C5444          .BYTE  128,98,92,84,68    ; 1/2    C6   A5   F5   A4   31
17B2 20544E4436          .BYTE  32,84,78,68,54     ; 1/8    F5   D5   A4   D4
17B7 2058484034          .BYTE  32,88,72,64,52     ; 1/8    G5   B4   G4   D@4
17BC 605C544A32          .BYTE  96,92,84,74,50     ; 3/8    A5   F5   C5   C4   32
17C1 205E544E32          .BYTE  32,94,84,78,50     ; 1/8    B@5  F5   D5   C4
17C6 4058524632          .BYTE  64,88,82,70,50     ; 1/4    G5   E5   B@4  C4
17CB 80544A443C          .BYTE  128,84,74,68,60    ; 1/2    F5   C5   A4   F4   33
17D0 00                  .BYTE  0                  ; END OF PIECE
```

A voltage controlled amplifier has as a minimum a signal input, a control input, and a signal output. The voltage at the control input determines the gain from the signal input to the signal output. In a typical setting, +8 V would correspond to unity (0 db) gain, with lower voltages decreasing the gain by 10 db per volt.

Many other voltage controlled devices have been developed during the approximately 12 year history of this field. In order to play music, the modules are first "patched" together with patch cords (like old style telephone switchboards) according to the desired sound characteristics. Manually operated control voltage sources such as potentiometers, joysticks and specialized organ-like keyboards are then manipulated by the player. The music is generally monotonic due to difficulties in the control elements (now being largely overcome). Multitrack tape recorders are universally utilized to produce the results heard on recordings such as Walter Carlos's *Switched on Bach*.

A useful computer interface to a synthesizer can be accomplished with nothing more than a handful of digital to analog and optionally analog to digital converters. The DACs would be used to generate control voltages under program control and the ADCs would allow operator input from the keyboard, for example, to be stored. Since control voltages vary slowly compared to the actual sound waveforms, real time control of a number of synthesizer modules is possible with the average microprocessor. Due to the large number of DACs required and the relatively slow speeds necessary, a multiplexing scheme using one DAC and a number of sample and hold amplifiers is appropriate. The home builder should be able to achieve costs as low as $2 per channel for a 32 channel, 12 bit unit capable of controlling a fairly large synthesizer.

The routing of patch cords can also be computerized. A matrix of reed relays or possibly CMOS bilateral switches interfaced to the computer might be used for this task. The patches used for some contemporary synthesizer sounds resemble the program patch boards of early computers and thus are difficult and time consuming to set up and verify. With computer controlled patching, a particular setup may be recalled

and set up in milliseconds, thus enhancing real time performance as well as reducing the need for a large number of different modules.

Other musical instruments may be interfaced as well. One well-published feat is an interface between a PDP-8 computer and a fair sized pipe organ. There are doubtless several interfaces to electronic organs in existence also. Even piano mechanisms can be activated, as noted elsewhere in this issue.

Recently, specialized music peripherals have appeared, usually oriented toward the S-100 (Altair) bus. In some cases these are digital equivalents of analog modules of similar function. For example, a variable frequency oscillator may be implemented using a divide-by-N counter driven by a crystal clock. The output frequency is determined by the value of N loaded into a register in the device, much as a control voltage affects a voltage controlled oscillator. Such an approach bypasses the frequency drift problems and interfacing expense of analog modules. The biggest advantage, however, is availability of advanced functions not feasible with analog modules.

One of these is a programmable waveform. A small memory in the peripheral holds the waveform (either as individual sample values or Fourier coefficients), which can be changed by writing in a new waveform under program control. Another advantage is that *time multiplexing* of the logic is usually possible. This means that one set of logic may simulate the function of several digital oscillators simultaneously, thus reducing the per oscillator cost substantially. Actually, such a digital oscillator may be nothing more than a hardware implementation of the PLAY routine mentioned earlier.

Digital/analog hybrids are also possible. The speech synthesizer module produced by Computalker Consultants, for example, combines a programmable oscillator, several programmable amplifiers and filters, white noise generator, and programmable switching on one board. Although designed for producing speech, its completely programmable nature gives it significant musical potential, particularly in vocals.

How do these various control techniques compare with the direct waveform computation techniques discussed earlier? A definite advantage of course is real time playing of the music. Another advantage is simpler programming, since sound generation has already been taken care of. However, the number of voices and complexity of subtle variations is directly related to the quantity of synthesizer modules available.

```
                        ;       WAVEFORM TABLE
                        ;       EXACTLY ONE PAGE LONG ON A PAGE BOUNDARY
                        ;       MAXIMUM VALUE OF AN ENTRY IS 63 DECIMAL OR 3F HEX TO AVOID
                        ;       OVERFLOW WHEN 4 VOICES ARE ADDED UP

0300                            .=      X'300           ; START WAVEFORM TABLE AT 0300

0300                    WAV1TB  =       .               ; VOICE 1 WAVEFORM TABLE
0300                    WAV2TB  =       .               ; VOICE 2 WAVEFORM TABLE
0300                    WAV3TB  =       .               ; VOICE 3 WAVEFORM TABLE
0300                    WAV4TB  =       .               ; VOICE 4 WAVEFORM TABLE
                                                        ; NOTE THAT ALL 4 VOICES USE THIS TABLE DUE
                                                        ; TO LACK OF RAM IN BASIC KIM-1

                        ;         FUNDAMENTAL AMPLITUDE  1.0 (REFERENCE)
                        ;         SECOND HARMONIC .5, IN PHASE WITH FUNDAMENTAL
                        ;         THIRD HARMONIC .5, 90 DEGREES LEADING PHASE

0300 3334353636                 .BYTE  X'33,X'34,X'35,X'36,X'36,X'37,X'38,X'39
0305 373839
0308 393A3A3B3B                 .BYTE  X'39,X'3A,X'3A,X'3B,X'3B,X'3B,X'3C,X'3C
030D 3B3C3D
0310 3C3C3C3C3C                 .BYTE  X'3C,X'3C,X'3C,X'3C,X'3C,X'3C,X'3C,X'3C
0315 3C3C3C
0318 3C3C3C3B3B                 .BYTE  X'3C,X'3C,X'3C,X'3B,X'3B,X'3B,X'3B,X'3B
031D 3B3B3B
0320 3A3A3A3A3A                 .BYTE  X'3A,X'3A,X'3A,X'3A,X'3A,X'3A,X'39,X'39
0325 3A3939
0328 3939393939                 .BYTE  X'39,X'39,X'39,X'39,X'39,X'39,X'39,X'39
032D 393939
0330 3A3A3A3A3A                 .BYTE  X'3A,X'3A,X'3A,X'3A,X'3A,X'3B,X'3B,X'3B
0335 3B3B3B
0338 3B3C3C3C3D                 .BYTE  X'3B,X'3C,X'3C,X'3C,X'3D,X'3D,X'3D,X'3D
033D 3D3D3D
0340 3E3E3E3E3F                 .BYTE  X'3E,X'3E,X'3E,X'3E,X'3F,X'3F,X'3F,X'3F
0345 3F3F3F
0348 3F3F3F3F3F                 .BYTE  X'3F,X'3F,X'3F,X'3F,X'3F,X'3F,X'3F,X'3F
034D 3F3F3F
0350 3E3E3E3D3D                 .BYTE  X'3E,X'3E,X'3E,X'3D,X'3D,X'3C,X'3C,X'3B
0355 3C3C3B
0358 3B3A393838                 .BYTE  X'3B,X'3A,X'39,X'38,X'38,X'37,X'36,X'35
035D 373635
0360 3433323130                 .BYTE  X'34,X'33,X'32,X'31,X'30,X'2F,X'2E,X'2D
0365 2F2E2D
0368 2C2B2A2928                 .BYTE  X'2C,X'2B,X'2A,X'29,X'28,X'27,X'26,X'25
036D 272625
0370 2423222121                 .BYTE  X'24,X'23,X'22,X'21,X'21,X'20,X'1F,X'1F
0375 201F1F
0378 1E1E1D1D1D                 .BYTE  X'1E,X'1E,X'1D,X'1D,X'1D,X'1D,X'1C,X'1C
037D 1D1C1C
0380 1C1C1D1D1D                 .BYTE  X'1C,X'1C,X'1D,X'1D,X'1D,X'1D,X'1D,X'1E
0385 1D1D1E
0388 1E1F1F2020                 .BYTE  X'1E,X'1F,X'1F,X'20,X'20,X'21,X'21,X'22
038D 212122
0390 2323242425                 .BYTE  X'23,X'23,X'24,X'24,X'25,X'26,X'26,X'27
0395 262627
0398 2828292929                 .BYTE  X'28,X'28,X'29,X'29,X'29,X'2A,X'2A,X'2B
039D 2A2A2B
03A0 2B2B2B2B2B                 .BYTE  X'2B,X'2B,X'2B,X'2B,X'2B,X'2B,X'2B,X'2A
03A5 2B2B2A
03A8 2A2A292928                 .BYTE  X'2A,X'2A,X'29,X'29,X'28,X'27,X'27,X'26
03AD 272726
03B0 2524232221                 .BYTE  X'25,X'24,X'23,X'22,X'21,X'20,X'1F,X'1D
03B5 201F1D
03B8 1C1B191817                 .BYTE  X'1C,X'1B,X'19,X'18,X'17,X'15,X'14,X'13
03BD 151413
03C0 11100F0D0C                 .BYTE  X'11,X'10,X'0F,X'0D,X'0C,X'0B,X'09,X'08
03C5 0B0908
03C8 0706050403                 .BYTE  X'07,X'06,X'05,X'04,X'03,X'03,X'02,X'01
03CD 030201
03D0 0100000000                 .BYTE  X'01,X'00,X'00,X'00,X'00,X'00,X'00,X'00
03D5 000000
03D8 0000010101                 .BYTE  X'00,X'00,X'01,X'01,X'01,X'02,X'03,X'04
03DD 020304
03E0 0506070809                 .BYTE  X'05,X'06,X'07,X'08,X'09,X'0B,X'0C,X'0D
03E5 0B0C0D
03E8 0F10121315                 .BYTE  X'0F,X'10,X'12,X'13,X'15,X'16,X'18,X'1A
03ED 16181A
03F0 1B1D1F2022                 .BYTE  X'1B,X'1D,X'1F,X'20,X'22,X'23,X'25,X'27
03F5 232527
03F8 282A2B2C2E                 .BYTE  X'28,X'2A,X'2B,X'2C,X'2E,X'2F,X'30,X'31
03FD 2F3031

                                .END
```

Table 5: This table is an encoding of the samples of the waveform used by the program in listing 2. The table is exactly one memory page long on a page boundary. The maximum value of any entry is decimal 63 or hexadecimal 3F to avoid overflow when all four voices are summed.

For example, if more voices are needed, either more modules must be purchased or a multitrack tape recording must be made, which then takes us out of the strict real time domain. On the other hand, a new voice in a direct synthesis system is nothing more than a few bytes added to some tables and a slightly lengthened execution time. Additionally, there may be effects that are simply not possible with currently available analog modules. With a direct synthesis system, one merely codes a new subroutine, assuming that an algorithm to produce the effect is known.

A separate problem for the experimenter is that a "critical mass" exists for serious work with a direct synthesis system. To achieve complexity significantly beyond the 4 voice example program described earlier, a high speed, large capacity mass storage system is needed. This means an IBM type digital tape drive or large hard surface disk drive; usually at least $3000 for a new drive less interface. Used 7 track tapes and 2311 type disks (7.5 megabytes) are often available for $500 and certainly provide a good start if the user can design his own interface. Synthesizer modules or peripheral boards, on the other hand, can be purchased one at a time as needed.

```
*   TOCCATA AND FUGUE IN D-MINOR      BACH
*
VOICE1 40,0,0,0,0,30,0,0,0,0,0,0,0,60,0        10      30,30
VOICE2 37,0,0,0,0,0,0,0,50,0,0,0,0,50,0        10      60,60
VOICE3 0,0,9,0,38,0,0,0,38,19,0,0,0,28,0       15      100,250
TEMPO 1/4=1200

        /-/

002 1A3,1/64; 2A2,1/64
    1A@3,1/64; 2A@2,1/64
    1A3,1/8; 2A2,1/8
    R,1/32
    1G3,1/647; 2G2,1/64.
    1F3,1/64; 2F2,1/64
    1E3,1/64; 2E2,1/64
    1D3,1/64; 2D3,1/64
    1C#3,1/32; 2C#2,1/32
    1D3,1/16; 2D2,1/16
    R,1/4
    3D2,1/1; R,1/4
    2C#3,1F2; R,1/16
    1E3,7/16; R,1/16
    1G3,7/16; R,1/16
    1B@3,5/16; R,1/16
    1C#4,4/16; R,1/16
    1E4,3/16

        /-/

140 1B@4,1/8; 1G4,1/8; 1E4,1/8; 2E3,1/8; 3C#3,1/8
    1E3,1/32
    1G3,1/32
    1B@3,1/32
    1C#4,1/32
    1B@4,1/8
    1B@4,1/8; 1G4,1/8; 1E4,1/8; 1C#4,1/8; 2E3,1/8; 3C#3,1/8
    1A4,1/8; 1F#4,1/8; 1D4,1/8; 2F#3,1/8; 3C3,1/8
TEMPO 1/4=950
    1D3,1/32
TEMPO 1/4=1050
    1A3,1/32
TEMPO 1/4=1150
    1D4,1/32
TEMPO 1/4=1200
    1F#4,1/32
    1A4,1/8
    1A4,3/8; 1F#4,1/8; 1D4,1/8; 2F#3,1/8; 3C3,1/8
141 1D4,1/2; 1B@3,1/2; 2G3,1/2; 3G2,1/4
    1G4,1/2; 3B@2,1/4
    1E4,1/4; 1C#4,1/4; 2B@3,1/4; 3E2,1/4
    1F4,1/4; 1D4,2/4; 2A3,1/4; 3F2,1/4
142 1E4,1/2; 2A3,1/2; 3A2,1/2; R,1/4
    1C4,2/4; R,1/4
    1D4,4/2; 2F3,1/4; 3B@2,1/4
    2B@3,1/4; 2G3,1/4; 3G2,1/4
143 2A3,3/2; 2F3,3/2; 3D3,3/2; 3D2,3/2
END
```

Listing 3: Bach's "Toccata and Fugue in D Minor" as encoded in NOTRAN, a music language developed by the author (NOTRAN stands for NOte TRANslation). The main function of the language is to transcribe organ music, but it will work equally well with other types of music. Program statements are used to encode duration, pitch, attack and decay rates, and loudness of each note.

Music Languages

Ultimately, software for controlling the sound generation process, whether it be direct or real time control, is the real frontier. The very generality of computer music synthesis means that many parameters and other information must be specified in order to produce meaningful music. One function of the software package is to convert "musical units of measure" into physical sound parameters such as conversion of tempo into time durations. Another part is a language for describing music in sufficient detail to realize the control power available from music synthesis without burdening the user with too much irrelevant or repetitious detail. With a good language, a good editor for the language, and real time (or nearly so) execution of the language, the music system becomes a powerful composition tool much as a text editing system aids writers in preparing manuscripts.

Music languages can take on two forms. One is a descriptive form. Music written in a descriptive language is analogous to a conventional score except that it has been coded in machine readable form. All information in the score necessary for proper performance of the piece is transcribed onto the computer score in a form that is meaningful to the user yet acceptable to the computer. Additional information is interspersed for control of tone color, tempo, subtle variations, and other parameters available to the computer synthesist.

A simple example of such a language is NOTRAN (NOte TRANslation) which was developed by the author several years ago for transcribing organ music. Listing 3 shows a portion of Bach's "Toccata and Fugue in D Minor" coded in NOTRAN. The basic thrust of the language was simplicity of instruction (to both the user and the interpreter program), rather than minimization of typing effort.

Briefly, the language consists of statements of one line each which are executed in straight line sequence as the music plays. If the statement starts with a keyword, it is

a specification statement; otherwise, it is a note statement. Specification statements simply set up parameters that influence the execution of succeeding note statements and take no time themselves.

A VOICE statement assigns the timbre described by its parameters to a voice number which is used in the note statements. In the example score, the first group of parameters describe the waveform in terms that are implementation dependent, such as harmonic amplitudes. The next, isolated parameter specifies the overall loudness of the voice in relation to other voices. The last pair of parameters specifies the attack and decay times respectively for notes using this voice. Depending on the particular implementation, other parameters may be added without limit. For example, vibrato might be described by a set of three additional parameters such as vibrato frequency, amplitude, and a delay from the beginning of a note to the start of vibrato.

A TEMPO statement relates note durations in standard fractional terms to real time in milliseconds. The effect of a tempo statement lasts until another is encountered. Although the implementation for which the example was written required a sequence of tempo statements to obtain a retard, there is no reason why an acceleration or a retard set of parameters could not be added.

Note statements consist of one or more note specifications and are indented four spaces (the measure numbers are treated as comments). Each note specification begins with a voice number followed by a note name consisting of a letter, optional sharp (#) or flat (@) sign, and an octave number. Thus C#4 is one half step above middle C. Following the comma separator is a duration fraction. Any fraction is acceptable, but conventional musical fractions are normally used. Following the duration are two optional modifiers. A period (.) indicates a "dotted" note which by convention extends the note's duration by 50%. An "S" specifies a staccato note which is played as just an attack and decay (as specified by the corresponding voice statement) without any steady state. The presence of a semicolon (;) after a note indicates that additional notes which are intended to be part of the same statement are present, possibly extending to succeeding lines.

The execution sequence of note statements can become a little tricky due to the fact that note durations in the statement may not all be equal. The rule is that all notes in the statement start simultaneously. When the shortest one has ended, the notes in the next statement are initiated, even though some in the previous statement may be still sounding. This could continue to any depth such as the case of a whole note in the bass against a series of sixteenth notes in the melody. The actual implementation, of course, limits the maximum number of simultaneous tones that may be built up.

Also available is a rest specification which can be used like a note specification. Its primary function is to provide silent space between note statements, but it may also be used to alter the "shortest note" decision when a note statement is scanned. If the rest is the shortest then the notes in the next statement are started when the rest elapses even though none of the current notes have ended. A use of this property may be seen in the last part of measure 2 where an arpeggio is simulated.

As can be seen, NOTRAN is best suited for describing conventional organ music, although it could be extended to cover a wider area as well. One such extension which has been experimented with but not fully implemented is percussion instruments. First a set of implementation dependent parameters was chosen to define a percussive sound, and then a PRCUS statement similar to the VOICE statement was added to the language. To initiate percussive sounds, specifications such as "P3,1/4" would be interspersed with the note specifications in note statements. The "3" would refer to percussive sound number 3 and the 1/4 would be a "duration" which would be optional. All percussive sounds in the same statement would start simultaneously with the regular notes.

A much more general music language is the well-known MUSIC V. It was designed to make maximum use of the flexibility afforded by direct waveform computation without overburdening the user. It is a massive program written in FORTRAN and clearly oriented toward large computers. Much significant computer music work has been done with MUSIC V, and it is indeed powerful. An excellent book is available which describes the language in detail and includes some background material on digital sound generation (see entry 1 in the list of references at the end of this article).

A different approach to music languages is a "generative" language which describes the *structure* of the music rather than the note by note details. In use, the structure is described by "loops," "subroutines," and "conditional branches" much as an algorithm is described by a computer language. The structure is "executed" to produce detailed statements in a conventional music language which is then played to produce sound. The intermediate step need not necessarily be visible to the user. One well

thought out system is described in reference 2. It was actually developed as a musicological analysis tool and so has no provisions for dynamics, timbre, etc. It could, however, be extended to include these factors. One easy way to implement such a language is to write a set of macros using a good minicomputer macroassembler.

Conclusion

By now it should be apparent that computer generated music is a broad, multidisciplinary field. People with a variety of talents can make significant contributions, even on a personal basis. In particular, clever system designers and language designers or implementers have wide open opportunities in this field. Finally, imaginative musicians are needed to realize the potential of the technique.

A Short Cut to a Singing KIM. . .

As his article was being finished by our production department, Hal Chamberlin notified us that he has completed the design of a board which accomplishes the digital to analog conversion and filtering functions described in this article. The board contains printed circuitry for an 8 bit digital to analog converter, low pass filter and power amplifier. Without components, the board may be purchased for $6; completely assembled and tested the price is $35. Orders should be mailed prepaid to Micro Technology Unlimited, 29 Mead St, Manchester NH 03104. In addition, a software package for the KIM-1 computer is available on cassette tape (KIM format) for $13 added to the price of the output board. A 7 inch 16 ohm speaker can be ordered for $5 prepaid, completing the required parts of a KIM's music system. ■

REFERENCES

1. Mathews, Max, *The Technology of Computer Music*, MIT Press, Cambridge MA, 1969. Contains a detailed description of MUSIC V, the high level music language.

2. Smoliar, Stephen, "A Parallel Processing Model of Musical Structures," PhD dissertation, Massachusetts Institute of Technology, September 1971.

3. Oppenheim, A and Schafer, R, *Digital Signal Processing,* Prentice-Hall, NJ, 1975.