

Simulation Of Musical Instruments

By Hal Chamberlin

The use of microcomputers for teaching, composing, transcribing and playing music is rapidly becoming a major application area. New synthesizer boards, music programs, and integrated systems with music capability are among the new products highlighted in the microcomputer field. It is now common for university and even high school music departments to acquire a quantity of microcomputers solely for musical purposes. It is even getting to the point where it is hard to find a microcomputer owner *without* some kind of music program, even if it only plays kazoo-like music through a built-in two-inch speaker.

The Computer Music Field

Since any complete discussion of microcomputer music is impossible within the confines of a magazine format, this article deals with a much narrower subject area. First, however, we need to characterize the field somewhat to see how the topics of musical instrument simulation and software digital synthesis fit in.

Computer music systems cover a broad range of sophistication, application, capability, sound quality and cost. At one extreme, we have the limited-range, tinny, one-voice, "gee whiz" type of system mentioned earlier that can be set up for a dollar's worth of parts and a program whose listing would not even fill a page.

At the other extreme, we have experimental computer music systems in some universities that have a range beyond human perception, quadraphonic sound quality exceeding that of the best recording equipment, virtually unlimited synthesis capabilities and practically infinite voice count at a cost (if measured by industrial standards) in the millions. Using a microcomputer, you can set up a system with reasonably wide range, good stereo sound quality, good synthesis flexibility and 32 voices for a few thousand dollars. The important point is that there is a definite need for systems addressing these extremes and many points in between.

The simple one-voice systems are certainly the most common and are fully adequate for teaching elementary music concepts as well as for impressing friends and neighbors. In fact, they are probably preferred for getting started because their very simplicity makes them easy to learn and use. Since only pitch and timing can be controlled, there are only two variables to worry about. Harmony, timbre, envelope and dynamics are either absent or predetermined.

Note that this type of music system is easily implemented either purely in software using timed loops, which toggle an output port bit, or through a combination of software and hardware where control bytes are sent to a simple divide-by-N counter which may even be part of the I/O interface

chip used by the computer. With this level of system, you either quickly outgrow it and move on or are content to file the program alongside the Lunar Lander and Star Trek cassettes.

The next step up is generally either a synthesizer board or an inexpensive eight-bit digital-to-analog converter. The synthesizer board is a set of several oscillators which at a minimum are programmable for pitch and amplitude. (There are a couple of very sophisticated single-voice synthesizer boards, but they are intended to be used in multiples.)

The simplest type of synthesizer board has three square-wave oscillators with pitch and amplitude registers for each and sometimes an overall volume control. Typically, these boards are implemented using programmable timer integrated circuits as the oscillators (normally intended for use in process-control-oriented microcomputers) and discrete circuitry for the volume control function.

Recently, General Instrument introduced a synthesizer chip that has

Hal Chamberlin is vice president of Research and Development for Micro Technology Unlimited, Box 12106, Raleigh, NC 27605. Active in electronic sound synthesis since 1966 and in computer music synthesis since 1970, he has authored numerous magazine articles and has recently published a book entitled Musical Applications of Microprocessors.

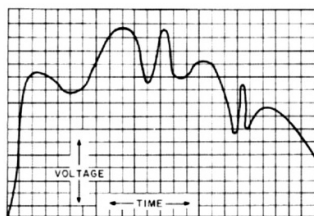


Fig. 1a. Desired audio waveform.

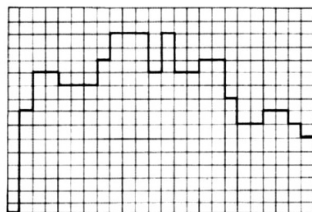


Fig. 1b. Raw DAC output.

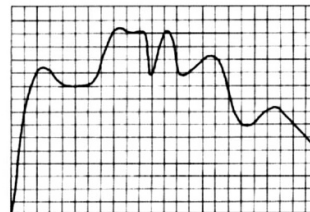


Fig. 1c. DAC output filtered by ideal low-pass filter.

the three oscillators and the volume control circuitry integrated on a single chip along with a noise generator useful for limited percussion effects. These chips, usually in trios for a total of nine voices, are appearing in the latest batch of synthesizer boards. Prices range from a little over \$100 to nearly \$300 with little connection between capability and price.

With these synthesizer boards, the computerist musician gains a great deal of flexibility, since he can play complex chords and control dynamics and tone envelopes. As a result, these boards are a great deal more difficult to master, although you can choose to ignore some of the variables initially.

There is one serious shortcoming, however: all synthesizer boards in this class produce square waves exclusively. (One three-voice board on the market has the capability of combining two of the voices into a single variable width rectangular wave, which increases the tonal variety somewhat.) Square waves have a rather sharp, yet hollow, sound that most closely resembles that of a kazoo. By suitable control of the amplitude envelope, you can produce continuous organ-like tones and percussive plucked-like tones, but the basic character of square waves remains.

With the proliferation of this type of board in recent months (and its constant demonstration at computer shows), the public may very well come to associate square wave sound with computers, just as a piano is associated with its own tone color. This would be unfortunate indeed, since the ultimate value added by computers in music is a wider range of timbres than any other instrument. Nevertheless, there is sufficient expressive power available so that the difference between a piece programmed by a novice and one programmed by an experienced musician is readily apparent.

Music systems based on these synthesizer boards seem to satisfy many users whose goal is to learn music, enjoy transcribing music into the computer and even perform simple composition. They typically will not satisfy a musician attempting to do serious performance work with the computer.

Much more sophisticated synthesizer boards with programmable waveforms are also available in the \$500 to \$2000 price range. These overcome the lack of tonal variety of the square wave boards by providing programmable waveforms, usually with provisions for a different waveform for each voice.

An important consideration that will be discussed later is whether the board allows dynamically variable waveforms; that is, the ability to smoothly alter the waveform while a note is being played with it. This is a requirement for many effects such as the "wah" of a muted trombone, and, as might be expected, the less expensive units do not provide for it. In either case, the tonal variety is far greater than the square wave units and is sufficient to satisfy many musicians as well as casual users.

On the other side of the hardware/software fence are music systems based on digital-to-analog converters (DAC). As we shall see later, a digital-to-analog converter simply translates numbers into voltages. A very rapid string of numbers produces a rapidly varying voltage; that is, an audio waveform.

In theory, appropriate software can calculate the necessary number sequence to produce literally any sound. The capabilities of a music system based on a digital-to-analog converter are determined solely by the sophistication of the software involved rather than the capabilities designed and frozen into a hardware synthesizer. DAC boards also tend to be less expensive. A good eight-bit DAC board sells for less than \$70,

while an experimental homebrew unit can be put together for half the price of a movie ticket.

The remainder of Part 1 will describe how sound generation software works in a DAC-based system.

Numbers to Sound

The fundamental principle behind digital sound synthesis is that a string of numbers from a computer program may be converted into a high-fidelity audio signal. As you might expect, the rate at which the numbers are supplied and the precision of the numbers both determine the fidelity of the resulting sound. In synthesis applications, fidelity's usual definition, i.e., faithfulness to the original, does not apply, since there is no original. Instead, fidelity is used to refer to the frequency range that can be produced and the relative freedom from undesired noise and distortion.

Fig. 1 shows how a DAC can produce a smooth audio waveform from a string of numbers and the errors involved. The grid in the figures represents time in the horizontal direction and voltage in the vertical direction. Fig. 1a shows a greatly magnified drawing of a small portion of a typical audio waveform. Notice that it wiggles and curves through the grid without regard for the grid. In Fig. 1b we have the raw output of a DAC being fed the string of numbers representing the waveform in Fig. 1a.

Each vertical grid line represents the point in time that the DAC receives a new number; thus, it stands to reason that the DAC output can only change up or down at vertical grid lines. Each horizontal grid line represents a possible numerical value that the DAC can receive. For example, an eight-bit DAC can only accept 256 (2^8) different numbers, so the complete grid for such a DAC would have 256 horizontal grid lines. As a result, the DAC output can only dwell at a horizontal grid line. Needless to say, the smoothly curved waveform of

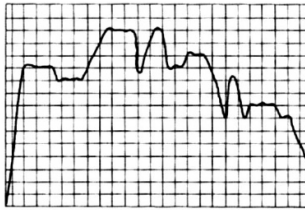


Fig. 1d. Filtered DAC output at twice the sample rate.

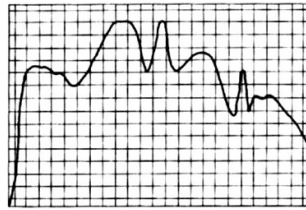


Fig. 1e. Filtered DAC output at twice the resolution and sample rate.

Fig. 1a is severely distorted in shape by this two-dimension quantization imposed by the DAC.

In Fig. 1c we have passed the raw DAC output through an ideal low-pass filter. Usually such a filter is described as allowing all frequencies below a certain cutoff frequency to pass through while blocking those above it. Here, however, we are using it to smooth the DAC output into something that more closely resembles Fig. 1a.

As you can see, the filter does a nice job of smoothing the curve between the vertical grid lines (that is, between new numbers from the computer). Closer examination, however, reveals that when the curve crosses a vertical grid line, it is also precisely on a horizontal grid line as well. Thus, since it is still constrained by the horizontal grid, there is still some error remaining. The only way to reduce this error is to make the horizontal grid lines denser by adding significant bits to the numbers and the DAC.

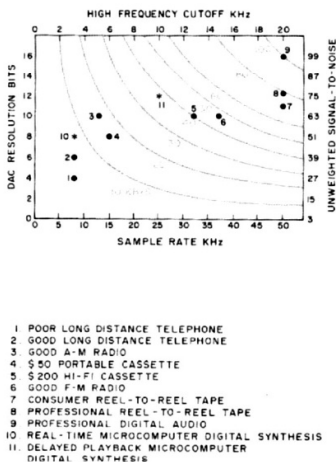


Fig. 2. Required DAC resolution, sample rate and data required for varying degrees of fidelity.

Comparing Fig. 1c with Fig. 1d, it should be apparent that the horizontal spacing of vertical grid lines determines the shortest waveform wiggle that can be reproduced. Since the horizontal direction is time, this is the same as saying that the horizontal spacing limits the highest audio frequency that can be reproduced. In particular, it takes at least two grid lines to reproduce one cycle of a sine waveform, so it follows that the highest audio frequency must be equal to or less than one-half of the rate at which numbers are fed to the DAC. Actually, you can go this high only when an ideal low-pass filter is used to do the smoothing.

With a practical low-cost filter, audio frequencies should be kept to 40 percent or less of the DAC update, or sample rate. If higher frequencies are attempted, the filter cannot properly smooth the waveform and an unpleasant distortion called alias distortion occurs. Note that there is no low-frequency limit when producing sound with a DAC; you can go clear down to dc if desired.

No matter how dense the vertical grid lines become, the waveform accuracy is always constrained by the horizontal grid line spacing. This constraint leads to background noise and distortion that cannot be filtered out.

Fig. 1e illustrates the effect of adding another bit of precision to the DAC and the numbers it receives. In terms of audio noise level, adding the bit reduces noise by six decibels, a significant but not dramatic amount. In most cases the noise itself is basically white, resembling somewhat the sound of ocean surf.

Fig. 2 shows how the fidelity of various combinations of sample rate (horizontal grid spacing) and DAC resolution (vertical grid spacing) compares with familiar audio devices. Also shown are contours of constant data rate (total kilobytes per second) to give an idea of required

system speed.

The important points are that frequency range and background noise level are independently adjustable system parameters and that greater fidelity is accompanied by a higher data rate. Note that it is considerably less expensive in terms of data rate to reduce the noise level than it is to increase the high-frequency limit. The two stars in Fig. 2 represent the two software digital music synthesis systems that will be discussed in this article.

Where the Numbers Come From

The real trick in a DAC-based music system, then, is to compute the string of numbers, or samples, representing the desired sound and then send it to the DAC at the required rate. In all of the cases that will be considered here, the sample rate will be constant because that assumption greatly simplifies the computations. Conversely, when the rate is assumed to be constant, it must be to rather close tolerances to avoid excessive jitter noise.

At this point you can choose to go in either of two directions. In real-time digital synthesis, the samples are computed at the rate required by the DAC and sent to it immediately. The advantage, of course, is that the sound is heard in its final form as the program is running. The disadvantage is that practical sample rates are relatively high, which means that a very efficient program using an uncomplicated synthesis technique running on a fast microcomputer is required.

The other choice is delayed playback digital synthesis, where the computed samples are first written onto a mass storage device at relatively low speed and then later reread and played through the DAC at the necessary high speed. The advantages here are that the synthesis program can be more accurate (and thus slower), any synthesis technique of any complexity can be utilized, and the higher sample rates and DAC resolutions necessary for high fidelity can be utilized. The main disadvantages are a rather long delay between program execution and audible results and the need for a large capacity, high-speed mass storage system.

It is also possible to combine the two philosophies—real time for composition and experimentation with the orchestration and delayed playback for a high-fidelity final result.

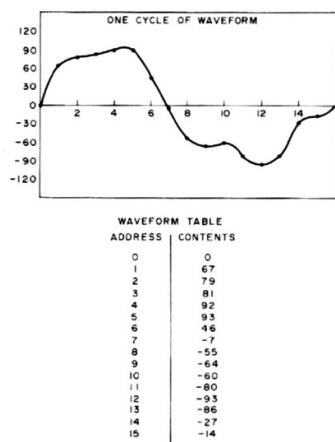


Fig. 3. Example waveform table.

The emphasis in this article will be on the real-time type of system, since that is of interest to most people at this time.

It should be noted that the software techniques that will be described actually perform the same functions as the hardware programmable waveform synthesizer boards mentioned earlier. In fact, a study of the history of computer music reveals that the techniques were developed first in software and then later implemented in hardware when it became practical to do so.

Let's begin by choosing a microprocessor, a sample rate and a DAC word size and then determine what can be done within those constraints.

The microprocessor will be a 6502 because of its usage in a large number of computers (PET, Apple II, Atari, KIM-1, SYM-1, AIM-65, OSI and Mattel, to name a few) and its effective high speed in this application (at a standard 1 MHz, approximately 60 percent faster than a 2 MHz 8080 or Z-80). About the lowest sample rate of interest is 8 kHz. This would allow audio frequencies up to a little beyond 3 kHz, which is just below the highest note on the piano keyboard. More important, it might limit the high harmonics of many instrumental sounds and thus give them a somewhat muffled character. Nevertheless, it is high enough to be useful.

Needless to say, the DAC word size will be eight bits because of the microprocessor chosen. This would give an audible but acceptably low background noise level with a DAC having true eight-bit linearity and a

good low-pass filter.

Using the 8 kHz sample rate means that the time between samples is a mere 125 μ s, or around 40 machine-language instructions per sample. Clearly you cannot write the program in BASIC, use the SIN function to compute samples, use floating-point arithmetic or even perform a simple multiply operation and get a sound sample computed in 125 μ s, much less several for multi-part harmony. Although there are simple (and thus fast) methods of computing sawtooth, triangle and square wave samples directly, only three different waveforms would not be very much variety.

The answer is to compute the waveform ahead of time and put the results in a waveform table. The music playing program then merely looks up the waveform samples in the table—an operation that takes almost no time on a 6502—and sends them to the DAC. Since the waveform is precomputed, even a very slow BASIC program is acceptable for computing it.

A waveform table is nothing more than a string of bytes in memory where each byte is a sample along the stored waveform. For simplicity, which means speed, waveform tables will be made 256 bytes long, or one memory page. The 256-byte content of the waveform table represents exactly one cycle of the stored waveform. Fig. 3 aids in understanding the relation between a waveform and its image in the table. For illustration purposes the table length is assumed

to be 16 bytes, but the same principles hold for the 256-byte length.

Now let's see how a music synthesis program might look up, or scan, the waveform table to produce sound. In its simplest form, scanning consists of reading the first entry, sending it to the DAC, reading the second entry, sending it to the DAC, etc., in a loop. When the end of the table is reached, it is necessary to wrap around to the beginning for the next cycle of the waveform.

If you did this at an 8 kHz rate with a 256 entry table, approximately 31 scans would be performed each second, which corresponds to 31 waveform cycles per second, a very low frequency note indeed. For higher frequency notes the scanning could be altered so that every other entry was read, which would give 62 Hz, every third for 93 Hz, etc. Although not intuitively obvious, skipping

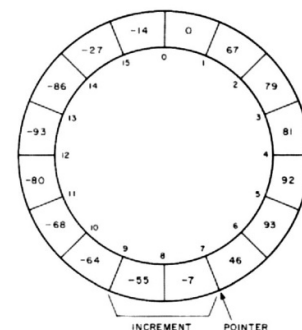


Fig. 4. Waveform table scanning.

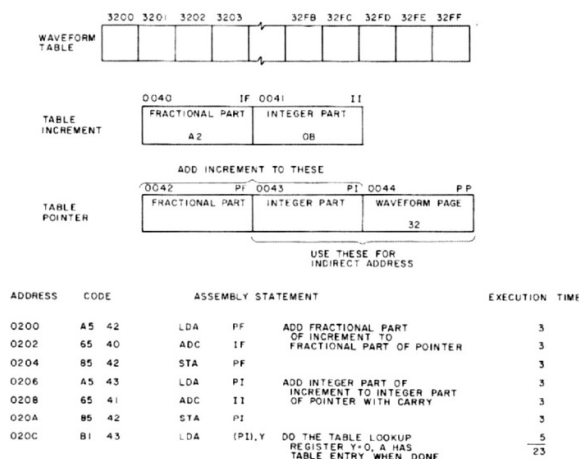


Fig. 5. Table scanning on the 6502.

waveform table entries does not contribute to distortion if the tabulated waveform conforms to certain rules that will be discussed later.

Fig. 4 aids in understanding the scanning process. Here the example 16-point waveform table has been bent into a circle, which is one way to view the wrap-around process mentioned earlier. The arrow represents the waveform table pointer, which contains the contents of a machine register or memory location. The bracket represents the value of the waveform table increment, which indicates how far the table pointer is advanced every 125 us sample period.

Thus, if the increment is one, the pointer will take on values of 0, 1, 2, ..., 14, 15, 0, ... (0-255 in real life) and give us a low note. If the increment is 3, the pointer will go through the sequence 0, 3, 6, 9, 12, 15, 2, ... and give us a three-times-higher note. Thus, the increment is proportional to the pitch of the synthesized tone. Note that in this case successive trips around the table are not exactly the same. Again, this does not lead to distortion if the waveform meets certain requirements.

Returning to the real case of a 256 point table, it is apparent that the frequency resolution of 31 Hz when using integral waveform table increments is not sufficient for most musical applications. What is needed is the ability to specify an increment with a fractional part such as 7.04 to produce a precise A below middle C. This is quite possible but requires that the waveform pointer also take on a fractional part, which leads to a problem. How should the table be read when the pointer says "read the 78.645th entry"?

A sensible answer would be to look at both the 78th and 79th entries and then interpolate between them. Unfortunately, even simple linear interpolation is fairly complex (requires a multiply), which means it is slow. For real-time digital synthesis on a microcomputer, we will be forced to ignore the fractional part of the pointer when reading the table but include it when adding the increment to compute the next value of the pointer. Taking this shortcut leads to a distortion called interpolation noise, which is significant but generally tolerable.

Now how might a program segment be set up to manipulate the pointer, increment and table to generate sample values for the DAC? Fig.

```

A00F DAC = X'A00F ; OUTPUT PORT ADDRESS WITH DAC
3000 WAV1TB = X'3000 ; WAVEFORM TABLE FOR VOICE 1
3100 WAV2TB = X'3100 ; WAVEFORM TABLE FOR VOICE 1
3200 WAV3TB = X'3200 ; WAVEFORM TABLE FOR VOICE 1
3300 WAV4TB = X'3300 ; WAVEFORM TABLE FOR VOICE 1

0000 . = 0 ; STORAGE STARTS AT PAGE 0 LOCATION 0

0000 00 V1PT: .BYTE 0 ; VOICE 1 WAVE POINTER, FRACTIONAL PART
0001 00 .BYTE 0 ; INTEGER PART
0002 30 .BYTE WAV1TB/256 ; WAVEFORM TABLE PAGE ADDRESS FOR VOICE 1
0003 00 V2PT: .BYTE 0 ; SAME AS ABOVE FOR VOICE 2
0004 00 .BYTE 0
0005 31 .BYTE WAV2TB/256
0006 00 V3PT: .BYTE 0 ; SAME AS ABOVE FOR VOICE 3
0007 00 .BYTE 0
0008 32 .BYTE WAV3TB/256
0009 00 V4PT: .BYTE 0 ; SAME AS ABOVE FOR VOICE 4
000A 00 .BYTE 0
000B 33 .BYTE WAV4TB/256

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 B6 TEMPO: .BYTE 182 ; TEMPO CONTROL VALUE, TYPICAL VALUE FOR
; 4:4 TIME, 60 BEATS PER MINUTE, DURATION
; BYTE = 48 (10) DESIGNATES A QUARTER NOTE

; 4 VOICE PLAY SUBROUTINE
; ENTER WITH VARIOUS TABLE POINTERS ALREADY SET UP
; LOOPS TEMPO*DUR TIMES
; = X'0300
0300 A000 PLAY: LDY #0 ; SET Y TO ZERO FOR STRAIGHT INDIRECT
0302 A615 LDX TEMPO ; SET X TO TEMPO COUNT
; COMPUTE AND OUTPUT A COMPOSITE SAMPLE
; CLEAR CARRY
0304 18 PLAY1: CLC ; ADD UP 4 VOICE SAMPLES
0305 B101 LDA (V1PT+1),Y ; USING INDIRECT ADDRESSING THROUGH VOICE
0307 7104 ADC (V2PT+1),Y ; POINTERS INTO WAVEFORM TABLES
0309 7107 ADC (V3PT+1),Y ; STRAIGHT INDIRECT WHEN Y INDEX = 0
030B 710A ADC (V4PT+1),Y ; SEND SUM TO DIGITAL-TO-ANALOG CONVERTER
030D 8D0FA0 STA DAC ; ADD INCREMENTS TO POINTERS FOR
0310 A500 LDA V1PT ; THE 4 VOICES
0312 650C ADC V1IN ; FIRST FRACTIONAL PART
0314 8500 STA V1PT
0316 A501 LDA V1PT+1
0318 650D ADC V1IN+1
031A 8501 STA V1PT+1 ; THEN INTEGER PART
031C A503 LDA V2PT ; VOICE 2

```

Listing 1. Core sound generation routine for organ-like music.

5 shows the arrangement of a waveform table, its pointer and its increment in memory. For illustration purposes, the waveform table is assumed to be in memory from 3200-32FF, which is page 32, while the pointer and increment are kept in memory page zero for fast access. The increment is a two-byte value with an integer byte and fraction byte as mentioned above. The decimal equivalent of the increment value shown is 11.633. The pointer is actually a three-byte value.

The most significant byte is the page number (32) of the waveform table and normally remains constant but can be changed to select a different waveform. The middle byte is the integer byte of the pointer into that table, while the least significant byte is the fractional part of the pointer.

Every sample period (125 us) the increment is double-precision added to the integer and fractional parts of the pointer, and the pointer is re-

placed with the result. Any overflow is simply ignored, since it is merely an indication of wrap-around from the end to the beginning of the waveform table. Actual table lookup is extremely simple in the 6502; you simply use the rightmost two bytes of the pointer (the waveform table page address and the integer part of the pointer) as the indirect address of an indirect load instruction. Thus, only one instruction is needed to look up in the waveform table. The 6502 machine-language code shown requires only 23 us to do all of this.

Since the 23 us figure is considerably less than the 125 us allowable, you can have several waveforms, pointers and increments for several simultaneous tones. There is enough time to handle four tones with some left over for housekeeping. You could also have fewer voices and a higher sample rate, or more and a lower rate.

There are two ways to combine the

four table-lookup values into a single eight-bit value for the DAC. One is to simply add them up and send the sum to the DAC, which is the equivalent of audio mixing. When this is done the waveform table values must have been adjusted when the table was computed to avoid overflow (which can lead to horrendous distortion) when the four voices are added up.

The other method is to immediately send each value to the DAC when it is found and let the low-pass filter smear them together, thus effecting mixing. One disadvantage of this approach is that the dwell time of each voice in the DAC must be the same or there will be differences in loudness among the voices. Another disadvantage is that certain DAC distortions are accentuated, although they are usually not significant at the eight-bit level. It is also a simple matter to have two DACs and direct two voices to each for an approximation to stereo.

Listing 1 shows the core sound generation routine used in a digital synthesis program first published in 1977. It is capable of generating four tones simultaneously, where each

tone can use a different waveform table. It uses the "add-em-up" technique of mixing the four voices into a single sample value for the DAC. A separate routine is expected to store the appropriate values in each of the four increments for the desired pitches and also set TEMPO and DUR for the desired duration of the chord.

Each time through the main loop takes 115 us and represents one sample period, thus the sample rate is 8.7 kHz. Also, each time through the loop decrements a copy of TEMPO, which is held in the X register. When X decrements to zero, it is restored from TEMPO, and DUR is decremented directly in memory. If DUR also decrements to zero, the chord is complete and a return to the setup routine is taken. Thus, the total chord duration is proportional to the product of TEMPO and DUR. This property makes it possible to change the speed of the music without recoding it.

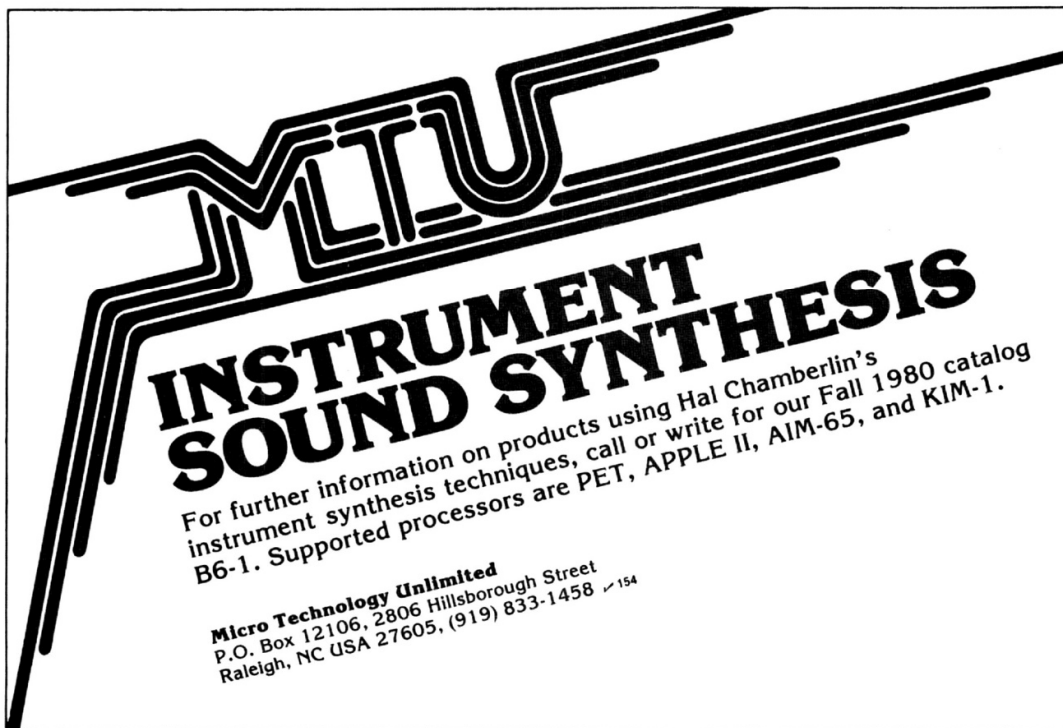
Note the presence of time-equalizing instructions at TIMWAS so that the loop time is the same whether or not register X decrements to zero.

This is necessary to eliminate jitter distortion mentioned earlier.

The setup routine would look at coded music in memory to determine what successive values of the four increments, DUR and possibly TEMPO should be to produce the desired music. Typically, music data would be set up in memory as a set of five bytes for each musical "event" (note or chord) in the piece. The first byte would be the duration, while the other four would represent the desired pitch of each of the four voices.

A note frequency table would be used to determine the proper two-byte value of the increment from the one-byte pitch code. This routine must also be as fast as possible because sound generation is stopped when it is in control. If the flow of samples is stopped for too long, an objectionable click between notes is introduced. See references for a further explanation of the setup routine.

Next month we will continue our discussion of synthesizing multiple tones using waveform table data, explore the capabilities of existing DAC software and examine some of the prospects for the future. ■



MTU

INSTRUMENT SOUND SYNTHESIS

For further information on products using Hal Chamberlin's instrument synthesis techniques, call or write for our Fall 1980 catalog B6-1. Supported processors are PET, APPLE II, AIM-65, and KIM-1.

Micro Technology Unlimited
P.O. Box 12106, 2806 Hillsborough Street
Raleigh, NC USA 27605, (919) 833-1458 ✓154