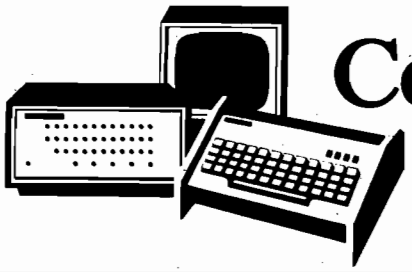

Computer Bits

By Hal Chamberlain

COMPUTER MUSIC

BEGINNING with this month, "Computer Bits" is a monthly feature of POPULAR ELECTRONICS. As a consequence, we are able to offer a two-part discussion of computer music—this month and in October.

Playing music with your computer can be a refreshing diversion, an impressive demonstration, or a mild obsession, depending on your inclination. We will discuss computer music performance first, with computer music composition to follow.

Some History. Elementary, monophonic music is just a series of tones with different frequencies and durations. Over the years, several methods have been devised by which a computer could either generate the tones directly or control tone-generating devices.

Perhaps the strangest method of output is one employed by an IBM-1401 computer program that played the line printer! Computer printers work much like a typewriter in that a metal type face strikes the paper under computer control. A pulse of sound is emitted when the paper is struck. In a 1401, the printer was interfaced in such a way that a program could control precisely when a print hammer fired. Proper relative timing among hammer strikes in the row of 120 hammers could actually produce a tone. Different frequency tones could be produced by changing the hammer timing. A rendition of "Anchors Aweigh" was heard by the author; and, while sounding rather raspy, it was in tune!

Another unexpected way to get a computer to produce tones involves the use of a common AM broadcast-band radio. Modern computer logic circuits generate pulses having risetimes of about 10 nanoseconds. These fast risetimes imply the existance of high-frequency harmonics that may fall into the radio's tuning range. Furthermore, the harmonic frequencies are separated from each other by the pulse repetition rate. An AM receiver interprets the harmonics within its passband as a carrier with upper and lower sidebands corresponding to audio modulation at the pulse repetition rate. Different instructions cause pulses to appear at different places within the computer. It is therefore possible to set up program loops that execute at controlled rates and thus produce tones of different frequencies. An interesting problem with this method is that there is no way to get the radio to "shut up" for a rest or a break between two identical notes.

A more predictable way to get audio output is to connect a speaker to an output port bit. A tone may be programmed by alternately setting the bit on and off with an accurately timed loop. This has an advantage since the speaker may be silenced when desired. One commercial computer (the LINC) even had a speaker complete with volume control connected to one bit of the accumulator as standard equipment! Some hobbyist systems have a speaker installed in their surplus keyboards. It is normally clicked to signal that a character has been accepted or beeped if an error is detected. Of course, it is also available as a simple "music peripheral."

Although the above methods make interesting demonstrations and do not require any specialized hardware, they are far from being suitable for serious music performance. Early in the development of electronic music, it became apparent that any audio signal waveform could be represented as a very rapid series of discrete voltage values. With a digital-to-analog converter connected to the computer,

Fig. 1. Example of a tone generation subroutine for the 8080.

```
                        *       ENTER WITH THE FREQUENCY PARAMETER IN C
                        *       ENTER WITH THE DURATION PARAMETER IN D & E                CLOCK
                        *       USES SPEAKER CONNECTED TO ANY BIT OF OUTPUT PORT          CYCLES

000:100 076 000     TONE    MVI  A,0          OUTPUT ZERO TO SPEAKER                     (7)
000:102 232 XXX             OUT  (port address)                                          (10)
000:104 101                 MOV  B,C          MOVE FREQ. PARAMETER TO                    (5)
000:105 005         DELAY1  DCR  B            B AND USE AS A DELAY COUNT                 (5)
000:106 302 105 000         JNZ  DELAY1       DELAY 15 X FREQ. PARAMETER CLOCK CYCLES    (10)
000:111 033                 DCX  D            DECREMENT DURATION PARAMETER               (5)
000:112 172                 MOV  A,D          TEST IF DECREMENTED TO ZERO                (5)
000:113 263                 ORA  E                                                       (4)
000:114 312 136 000         JZ   RETURN       GO RETURN IF SO, CONTINUE IF NOT           (10)
000:117 076 377             MVI  A,377Q       OUTPUT ONE TO SPEAKER                      (7)
000:121 323 XXX             OUT  (port address)                                          (10)
000:123 101                 MOV  B,C          DELAY AS ABOVE                             (5)
000:124 005         DELAY2  DCR  B                                                       (5)
000:125 302 124 000         JNZ  DELAY2                                                  (10)
000:130 033                 DCX  D            DECREMENT AND TEST DURATION AS ABOVE       (5)
000:131 172                 MOV  A,D                                                     (5)
000:132 263                 ORA  E                                                       (4)
000:133 302 100 000         JNZ  TONE         LOOP BACK FOR ANOTHER SQUARE WAVE CYCLE    (10)
000:136 311         RETURN  RET               RETURN TO MAIN PROGRAM
```

a series of numbers could be converted into the series of voltages and thus an audio signal. Waveforms could then be computed directly and literally any possible sound or tone produced. Some very impressive and musically important results have been obtained in this manner. Although this method has perfect generality, the number of computations needed for even a simple piece is staggering. Typically, hours of computer time are required for minutes or even seconds of musical output.

The newest method of musical output is to interface a computer to a sound synthesizer. Units such as a Moog or ARP or some of the circuits described by Don Lancaster in this magazine can be used. The amplitude, frequency, and spectrum of tones produced by the synthesizer are set with control voltages. Contrary to actual sound waveforms, control voltages typically change slowly. This greatly reduces the computational load on the computer. With only $50 to $100 worth of components, a multiplexed digital-to-analog converter with a dozen or more control voltage outputs can be built. Ordinary patch cords may then be used to connect the control voltages to the synthesizer.

Timed Loop Techniques. Let's look at the timed-loop technique of producing single tones with a computer. The two most important elements of music are pitch and rhythm. Curiously, these are the only elements that can be easily controlled in a timed-loop music program. Pitch is determined by the frequency of a tone in cycles per second (Hertz) and rhythm by the relative durations of individual tones in seconds. The computer used must have a definite clock cycle time, and most hobby computers on the market use a crystal clock which gives a very stable and accurate cycle rate. A few of the "trainer" kits may use an RC clock which is not nearly as good, but can still be used to illustrate the principles.

Fig. 2. Table of musical notes.

Note	Frequency Hertz	Period Microseconds
C (middle)	261.62	3822.3
C#	277.18	3607.8
D	293.66	3405.3
D#	311.13	3214.2
E	329.63	3033.8
F	349.23	2863.5
F#	369.99	2702.8
G	391.99	2551.1
G#	415.30	2407.9
A	440.00	2272.8
A#	466.16	2145.2
B	493.88	2024.8
C	523.25	1911.2
C#	554.37	1803.9
D	587.33	1702.7
D#	622.25	1607.1
E	659.26	1516.9
F	698.46	1431.8
F#	739.99	1351.4
G	783.99	1275.6
G#	830.61	1204.0
A	880.00	1136.4
A#	932.33	1072.6
B	987.77	1012.4
C	1046.5	955.58

The basic component of a timed-loop music program is the tone-generation subroutine. The tone frequency and duration are passed to the subroutine as two arguments. The main body of the subroutine consists of two "nested" loops. The frequency argument determines how many times the inner loop is executed. The duration argument determines how many times the outer loop is executed. Note that the frequency argument is actually proportional to the period of the wave and the duration is actually the number of cycles of the wave to be played before returning for the next note.

A tone generation subroutine for an 8080 microprocessor is shown in Fig. 1. This routine generates an absolutely symmetrical square wave and uses two inner loops of identical length to accomplish it. To determine what the frequency argument value should be, it is necessary to count up the clock cycles used by the inner wait loop and by the "overhead" instructions outside of the loop. The overhead is found to be 46 cycles and the wait loop is 15 cycles times the value of the frequency parameter. Assuming that the 8080 runs at full speed (0.5 microsecond clock cycle) with no memory waits, the time for a half squarewave cycle is 23+7.5N microseconds, where N is the frequency parameter. A full cycle time is twice as long. The frequency is, of course, the reciprocal of the full cycle period. Using the routine shown, N must be between 1 and 255; if zero is given, it is interpreted as 256. Thus, the lowest possible frequency is 500,000/(23+7.5×256) Hz. or 257.33 Hz. The table in Fig. 2 may be used to determine the proper value of N for any note.

The duration parameter is actually a count of half cycles to be played before returning. As a result, the argument value is a function of both the duration in seconds and the note played. Thus M=2TF, where M is the argument value, T is the time in seconds, F is the frequency in Hertz, and the factor of 2 accounts for both halves of the square wave. Note that the routine of Fig. 1 uses a double precision duration argument to provide for long notes.

Fig. 3. Basic specification for music language NOTRAN.

1. **TEMPO statement**
   Example: TEMPO ¼=500
   TEMPO is keyword identifier.
   ¼ refers to a "quarter note."
   500 is duration of quarter note in milliseconds.
2. **Note statement**
   Example: 1C#4,⅛
   1 refers to voice 1 (optional).
   C is note name, # following is sharp, @ is flat.
   4 is octave number, C4 is middle C.
   ⅛ specifies an eighth note.
3. **Rest statement**
   Example: R,½
   R signifies rest
   ½ means rest duration equivalent of half note.
4. **For monophonic music,** one statement per line

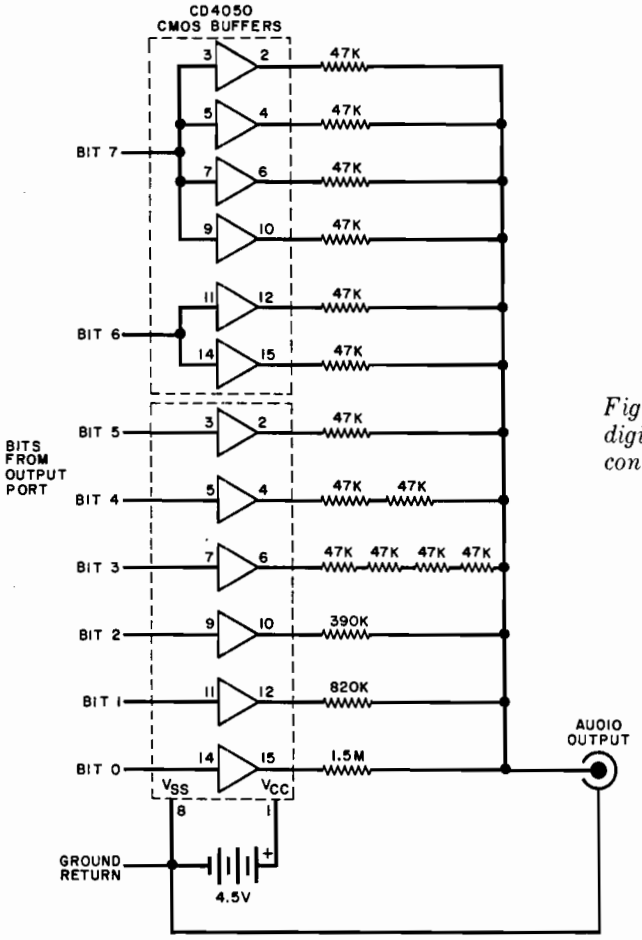

Fig. 4. Simple 8-bit digital-to-analog converter.

Using the subroutine in a music program is simply a matter of calling it with the right frequency and duration arguments for each note to be played. Care must be taken to insert some silence between notes of the same frequency or else they will run together. A simple-minded program might fetch these from a hand-prepared list stored in memory. A more challenging approach is to write a program that accepts statements written in a simple "music language" and computes the tone-generation arguments (Fig. 3). Then your musically inclined wife, for example, may perpare scores for the computer herself.

Improving Basic Timed Loop Techniques. For better sounding music, a number of refinements can be made to the basic technique. It is possible to get some limited change of tone color by changing the duty cycle of the wave. This may be accomplished by inserting some "no operation" instructions in one of the wait loops. Unfortunately, the frequency arguments will have to be recomputed. Good duty cycle values to try are ⅓ and ¼, since harmonics divisable by 3 and 4, respectively, will be missing from the spectra.

Another interesting experiment is to improve the frequency accuracy of the routine. As written, the half-cycle time is in increments of 7.5 microseconds. This results in significant frequency errors, particularly for high notes. By inserting additional selected overhead instructions, it is possible to be as accurate as 0.5 microsecond. This might be accomplished by writing 15 copies of the routine with extra instructions added in increments of one clock cycle. Obtaining the right note would then be a combination of calling the proper routine with the proper frequency argument.

The computer-generated tones can be made more natural sounding if an amplitude envelope is added to the notes. All this amounts to is changing the volume in a controlled manner over the duration of the note. Additionally, there would then be some control over the dynamics (changes in loudness during a piece) of the music. At this point, some specialized hardware must be added.

The circuit in Fig. 4 may be added to an output port. This is actually a very simple 8-bit digital-to-analog converter. Best results are obtained if the 47k resistors are 5% carbon film types from the same batch. Several mail-order firms now offer this type of resistor. A battery is used to power the circuit so that system noise will be isolated. Any type of audio amplifier can be connected to the circuit's output.

The dc output voltage of this circuit is 0.0176N volts, where N is the binary number last sent to the output port. Note, in the tone generation subroutine, that 0 and 255 (decimal) were alternately sent to the output port for a square wave. If this circuit was connected to the output port, the output voltage would alternately switch between 0 volts and +4.5 volts. If instead, 0 and 128 were alternately sent out, then the wave would switch between 0 volts and about 2.26 volts. This is only about one-half of the previous amplitude so the tone would not be as loud. Thus, amplitude control may be exercised by simply changing the values sent to the output port by the tone-generation subroutine.

Unfortunately, programming amplitude envelopes during a tone without upsetting the precise loop timing is a bit tricky. As the level of sophistication increases, a point is reached where the effort needed to preserve the loop timing is excessive. At this point specialized hardware is needed to relieve the computer of the actual tone generation task.

Experiments with Other Sounds. Other sounds can also be easily generated by a computer. For example, a high-speed stream of random bits will provide a good source of white noise. An 8080 program to generate white noise is given in Fig. 5. Random bits are generated by simulating a 16-stage shift register with appropriate feedback. White noise might be used in a rhythm program to approximate the sound of a snare drum.

The program in Fig. 6 produces some really "way-out" sounds. One should first try it and listen to each of the 16 possible output bits. Then study the program and see if you can figure out what it is actually doing. Finally, try to explain the weird results near the end of each repetition.

Many other interesting experiments can be performed with the simple digital-to-analog converter described earlier. Some of these will be described next time. ◈

```
                *       CONNECT SPEAKER TO ANY BIT OF OUTPUT PORT

000:000 021 000 000  WHITEN  LXI  D,0            SET D & E TO ZERO
000:003 041 001 000          LXI  H,1            RET SHIFT REG.  IN H & L NON-ZERO
000:006 174          LOOP    MOV  A,H            OUTPUT UPPER 8 BITS OF SHIFT REGISTER
000:007 323 XXX              OUT  (port address) AS A SET OF RANDOM BITS
000:011 017                  RRC                 FORM EXCLUSIVE-OR OF SHIFT REGISTER
000:012 254                  XRA  H              BITS 15,14,12, AND 3 IN BIT 0 OF A
000:013 017                  RRC
000:014 017                  RRC
000:015 254                  XRA  H
000:016 017                  RRC
000:017 255                  XRA  L
000:020 017                  RRC
000:021 017                  RRC
000:022 017                  RRC
000:023 346 001              ANI  1              ISOLATE BIT 0 OF A
000:025 051                  DAD  H              SHIFT SHIFT REGISTER LEFT 1
000:026 137                  MOV  E,A            AND BRING BIT 0 OF A INTO
000:027 031                  DAD  D              VACATED SHIFT REGISTER BIT 0
000:030 303 006 000          JMP  LOOP           LOOP
```

Fig. 5. White noise generation routine for 8080

```
                *       CONNECT SPEAKER TO ANY BIT OF OUTPUT PORT
                *       A DIFFERENT SOUND WILL BE HEARD AT EACH BIT

000:000 021 001 000  WEIRD   LXI  D,1            INITIALIZE
000:003 041 000 000          LXI  H,0
000:006 175          LOOP    MOV  A,L            CHANGE TO:  MOV A,H FOR 8 MORE SOUNDS
000:007 323 XXX              OUT  (port address) OUTPUT TO SPEAKER
000:011 031                  DAD  D
000:012 322 006 000          JNC  LOOP
000:015 023                  INX  D
000:016 172                  MOV  A,D
000:017 263                  ORA  E
000:020 302 006 000          JNZ  LOOP
000:023 023                  INX  D
000:024 303 006 000          JMP  LOOP
```

Fig. 6. Weird sound generation routine for the 8080.