A FREQUENCY MODULATION MUSIC SYSTEM

BY

ROBERT MICHAEL KAMINSKY

B.S., Rose-Hulman Institute of Technology, 1980

THESIS

Submitted in partial fulfillment of the requirements
for the degree of Master of Science in Electrical Engineering
in the Graduate College of the
University of Illinois at Urbana-Champaign, 1981

Urbana, Illinois
I would like to express my deepest appreciation to my thesis advisors, Prof. Ricardo Uribe and Prof. William Jenkins, for their guidance on this project and helpful suggestions in the writing of this manuscript.

Special thanks to Jim Graf for explaining the use of some of the lab equipment, Shaw Moldauer and Glenn Poole, for their suggestions concerning the Processor Board, and Prof. Ricardo Uribe, for his patience and helpfulness throughout the course of this project.
# TABLE OF CONTENTS

1. **INTRODUCTION** .................................................................................. 1
   1.1 Design Overview ........................................................................... 3
   1.2 Brief Historical Background ..................................................... 4

2. **THEORETICAL CONCEPTS** ................................................................. 6
   2.1 Static Spectra .............................................................................. 6
   2.2 Dynamic Spectra ......................................................................... 9
     2.2.1 Mathematical description .................................................. 9
     2.2.2 Example ............................................................................... 13

3. **SYSTEM DESIGN** ............................................................................... 16
   3.1 Frequency Generators ................................................................. 17
   3.2 Envelope Generators .................................................................. 19
   3.3 Multiplier Circuit ........................................................................ 21
   3.4 Digital-to-Analog Conversion ................................................... 21

4. **HARDWARE DESIGN** ...................................................................... 24
   4.1 Register Length ........................................................................... 24
     4.1.1 Frequency generator circuit ............................................. 25
     4.1.2 Envelope generator circuit .............................................. 26
   4.2 Pipeline Clock .............................................................................. 29
   4.3 Sine Wave Table .......................................................................... 29
   4.4 Multiplier Circuit ......................................................................... 31

5. **SYSTEM SOFTWARE** .......................................................................... 32
   5.1 Test Program ................................................................................ 33
<table>
<thead>
<tr>
<th>Figure</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.1</td>
<td>THE ADSL MUSIC SYSTEM</td>
<td>2</td>
</tr>
<tr>
<td>2.1</td>
<td>SINE WAVE SPECTRUM</td>
<td>7</td>
</tr>
<tr>
<td>2.2</td>
<td>SQUARE WAVE SPECTRUM</td>
<td>7</td>
</tr>
<tr>
<td>2.3</td>
<td>SPECTRUM OF SAWTOOTH AND SQUARE WAVE COMBINATION</td>
<td>8</td>
</tr>
<tr>
<td>2.4</td>
<td>ROTATING VECTOR REPRESENTATION OF FM SIGNAL</td>
<td>11</td>
</tr>
<tr>
<td>2.5</td>
<td>FM SPECTRUM (M=1)</td>
<td>14</td>
</tr>
<tr>
<td>2.6</td>
<td>FM SPECTRUM (M=3)</td>
<td>14</td>
</tr>
<tr>
<td>3.1</td>
<td>FREQUENCY GENERATOR BLOCK DIAGRAM</td>
<td>18</td>
</tr>
<tr>
<td>3.2</td>
<td>ENVELOPE GENERATOR BLOCK DIAGRAM</td>
<td>20</td>
</tr>
<tr>
<td>3.3</td>
<td>FMMS BLOCK DIAGRAM</td>
<td>23</td>
</tr>
<tr>
<td>4.1</td>
<td>MODEL OF MULTIPLIER QUANTIZATION NOISE</td>
<td>27</td>
</tr>
<tr>
<td>6.1</td>
<td>MODIFIED INCREMENT REGISTER FOR HANDLING MULTIVOICE</td>
<td>40</td>
</tr>
<tr>
<td>6.2</td>
<td>MODIFIED OUTPUT SECTION FOR HANDLING MULTIVOICE</td>
<td>41</td>
</tr>
<tr>
<td>A1</td>
<td>COMPONENT LAYOUT FOR FREQUENCY GENERATOR BOARD</td>
<td>43</td>
</tr>
<tr>
<td>A2</td>
<td>MPA CIRCUIT</td>
<td>44</td>
</tr>
<tr>
<td>A3</td>
<td>CMPA CIRCUIT</td>
<td>45</td>
</tr>
<tr>
<td>A4</td>
<td>COMPONENT LAYOUT FOR ENVELOPE GENERATOR BOARD</td>
<td>46</td>
</tr>
<tr>
<td>A5</td>
<td>PENV CIRCUIT</td>
<td>47</td>
</tr>
<tr>
<td>A6</td>
<td>MCND SELECTOR CIRCUIT</td>
<td>48</td>
</tr>
<tr>
<td>A7</td>
<td>AENV CIRCUIT</td>
<td>49</td>
</tr>
<tr>
<td>A8</td>
<td>COMPONENT LAYOUT FOR MULTIPLIER BOARD</td>
<td>50</td>
</tr>
<tr>
<td>A9</td>
<td>MULTIPLIER SELECTOR CIRCUIT</td>
<td>51</td>
</tr>
<tr>
<td>A10</td>
<td>MULTIPLIER SYNCHRONIZATION CIRCUIT</td>
<td>52</td>
</tr>
<tr>
<td>A11</td>
<td>MULTIPLIER &amp; OUTPUT CIRCUIT</td>
<td>53</td>
</tr>
<tr>
<td>Figure</td>
<td>Description</td>
<td>Page</td>
</tr>
<tr>
<td>----------</td>
<td>--------------------------------------------------</td>
<td>------</td>
</tr>
<tr>
<td>A12</td>
<td>COMPONENT LAYOUT FOR PROCESSOR BOARD</td>
<td>54</td>
</tr>
<tr>
<td>A13</td>
<td>CLOCK GENERATOR CIRCUIT</td>
<td>55</td>
</tr>
<tr>
<td>A14</td>
<td>CENTRAL PROCESSING UNIT</td>
<td>56</td>
</tr>
<tr>
<td>A15</td>
<td>MEMORY &amp; I/O</td>
<td>57</td>
</tr>
<tr>
<td>A16</td>
<td>SERIAL I/O &amp; KEYBOARD PORT</td>
<td>58</td>
</tr>
<tr>
<td>B1</td>
<td>SYSTEM TIMING DIAGRAM</td>
<td>60</td>
</tr>
</tbody>
</table>
LIST OF TABLES

Table 4.1 -- ACCURACIES OF VARIOUS REGISTER LENGTHS ................. 26
Table 4.2 -- MULTIPLIER OUTPUT SNR FOR VARIOUS INPUT REGISTER LENGTHS ................................................. 28
Table 5.1 -- TEST PROGRAM PARAMETERS ........................................ 36
CHAPTER 1

INTRODUCTION

This thesis describes the FMMS (Frequency Modulation Music System) which uses frequency modulation (FM) techniques to produce sounds which have dynamic frequency spectra. By using FM, a multitude of musical timbres can be produced. The system was designed and built in the ADSL (Advanced Digital Systems Laboratory) and integrates into the laboratory's Music System. This system is controlled by an 8085 CPU (Central Processing Unit) and interfaces to an organ keyboard, an amplifier and speakers, disk memory and several different music synthesizers. Figure 1.1 shows a block diagram representation of this arrangement.

The text is organized to first acquaint the reader with some of the theory behind FM and then goes on to present the considerations which were made in the actual designing of the system. Chapter two describes the mathematical theory behind frequency modulation, using spectral
Figure 1.1 THE ADSL MUSIC SYSTEM
analysis to explain how a tone's timbre (sound character) can be enriched. Chapter three describes the hardware block structure of the FMMS, whereas chapter four goes into the details concerning the hardware design considerations. Chapter five is concerned with the system software, and in chapter six, some ideas are given on how to expand the FMMS to handle multivoice, followed by some concluding remarks.

1.1 Design Overview

The FMMS is chiefly an experimental project. The primary goal is to allow the user enough flexibility to create a variety of sounds presently unobtainable with the ADSL Music System.

The FMMS consists of four vector boards which are mounted in a rack in the ADSL Music System cabinet. The first three boards manipulate the data sent to them by the fourth board (the Processor Board). The hardware consists of pipeline registers and adders, a sine wave look-up table, a multiplier circuit and an output circuit consisting of a DAC (Digital-to-Analog Converter) and a low pass filter.

The data sent to the FMMS system hardware is controlled by an 8085 CPU. The CPU basically waits for interrupts from the keyboard port. When it senses that a key has been pressed or released, it sends out the appropriate data at the proper times to each of five generator registers.
1.2 Brief Historical Background

FM synthesis of music is a technique which is only about ten years old. Synthesizing music by electronic means dates back to the early 1900's with the development of Thadeous Cahill's Telharmonium. This was a huge machine which was driven by alternators. In 1920, the Theremin was introduced by USSR's Leon Theremin. It was an antennae synthesizer which used the capacitance from one's hand to alter the beat frequency from dual oscillators. The first electronic music studios were introduced in the early 1950's.

With the development of the computer, new ways of synthesizing music seemed limitless. Max Mathews of Bell Labs was the first to use digital computers to produce music back in 1960. By the early 1970's, John Chowning demonstrated the effectiveness of frequency modulation techniques in simulating "natural instrument tones." The late 1970's marked the advent of fabricating music synthesizers on a single integrated circuit chip [1].

Presently, three means of synthesizing music are in vogue. Additive synthesis, which generates a sine wave for each harmonic and adds them all together. Non-linear synthesis, which "processes" an input sine wave into various harmonic frequencies. The processor essentially transforms the input into a power series expansion whose coefficients depend on the input amplitude [1]. Finally, FM, which controls the
evolution of the spectrum rather than controlling each individual frequency component.
CHAPTER 2
THEORETICAL CONCEPTS

2.1 Static Spectra

Many attempts have been made at producing tones with interesting timbres. Musical Engineers have tried using combinations of triangle, square, sine and sawtooth waveforms in an effort to simulate a timbre with more life and zest than the usual electronic sounding synthesizers which use square or sine waves exclusively. The problem with these attempts is that regardless of how one combines these various waveforms, the spectrum it produces will be static. For example, let us say one wants to produce an interesting tone with a fundamental frequency of 100Hz. By using a sine wave, the spectrum simply includes the fundamental at 100Hz as shown in Figure 2.1. Although simple, this tone lacks character. Next, one could use a square wave, producing the spectrum diagrammed in Figure 2.2. Square waves sound nasally and
Figure 2.1 SINE WAVE SPECTRUM

Figure 2.2 SQUARE WAVE SPECTRUM
somewhat piercing. In Figure 2.3, the previous spectrum is added to that of a sawtooth wave (with fundamental frequency of 100Hz).

![Figure 2.3 Spectrum of Sawtooth and Square Wave Combination](image)

By adding these components vectorially one notices that every other odd harmonic has been eliminated. This spectrum is becoming broader (amplitudes of harmonics are increasing). This tends to add more brightness or color to the tone.

The purpose of the previous illustrations is to give the reader some idea of what is meant by a broad spectrum and how a note's timbre can be enriched by broadening the spectrum.
Notice that in the last two illustrations, although the spectrum is complex, the amplitudes of all the frequency components remain constant throughout the duration of the note. It is for this reason that these techniques have failed in any attempts to simulate common musical instruments (e.g. brass, woodwinds, percussion etc.), the ultimate test of a music system's capabilities [2].

2.2 Dynamic Spectra

The FM equation allows dynamic control of the spectrum by varying a parameter called the modulation index. If the timbre of common instrument tones depended upon the relative magnitudes of each frequency component, the FM approach would probably be ineffective. However, the results achieved by John Chowning and others have shown that the timbre is more dependent on the evolution of the spectrum as a whole rather than the evolution of individual components [3].

2.2.1 Mathematical description

To see how a dynamic spectrum is realized using linear frequency modulation, the general FM equation will first be derived. To produce an FM signal, one superimposes an AC waveform on a DC waveform and uses this signal to control a variable oscillator whose instantaneous output frequency is proportional to the input signal. Thus:
\[ f_{\text{inst}} = K_{d,c} + K_{a,c} \]
\[ = f_c + f_d \cos (2\pi f_m t) \]

where the modulating frequency, \( f_m \), is the frequency of the AC signal. The carrier frequency, \( f_c \), is proportional to the DC signal, and \( f_d \) is the maximum amount by which the output frequency deviates from the carrier frequency. The output signal could be thought of as the vertical projection of a vector rotating on the unit circle as illustrated in Figure 2.4. The function \( A(t) \) is the instantaneous phase of the vector \( S(t) \). Since the time derivative of the phase is the instantaneous frequency, an expression for the instantaneous phase can be obtained by integrating equation (1):

\[ A(t) = 2\pi \int_0^t [f_c + f_d \cos (2\pi f_m t')] \, dt' \]

\[ = 2\pi f_c t' + (f_d / f_m) \sin (2\pi f_m t) \]

Therefore the output signal \( S(t) \) is:

\[ S(t) = \sin [2\pi f_c t' + (f_d / f_m) \sin (2\pi f_m t)] \]

The coefficient \( f_d / f_m \) is known as the modulation index (M). Equation (1) can now be rewritten in the following form:

\[ f(n) = f_c + (M f_d) \cos (2\pi f_m n T) \]

Where equation (1) has been changed from a continuous to a discrete frequency whose sampling period is denoted by \( T \). The output signal, therefore, can be completely specified by the three parameters: \( f_c \), \( f_m \) and \( M \).
Figure 2.4 ROTATING VECTOR REPRESENTATION OF FM SIGNAL
Now by referring back to equation (3) we can begin our realization of the dynamic spectral behavior that is characteristic of frequency modulation.

Using a trigonometric identity:

\[
S(t) = \sin (2\pi f_c t) \cos [M \sin (2\pi f_m t)] \\
+ \cos (2\pi f_c t) \sin [M \sin (2\pi f_m t)]
\]

In order to simplify this expression still further one needs to utilize the "Bessel Functions" which define the sine of a sine and cosine of a sine in terms of Bessel polynomials. The identities are:

\[
\sin [M \sin(x)] = 2[J_1(M)\sin(x) + J_3(M)\sin(3x) + \ldots] \quad (6)
\]
\[
\cos [M \sin(x)] = J_0(M) + 2[J_2(M)\cos(2x) + J_4(M)\cos(4x) + \ldots] \quad (7)
\]

Thus:

\[
S(t) = \sin(2\pi f_c t) \ J_0(M) + 2 [J_2(M)\cos(4\pi f_m t) + \ldots] \\
+ \cos(2\pi f_c t) \ 2[J_1(M)\sin(2\pi f_m t) + J_3(M)\sin(6\pi f_m t) + \ldots]
\]

Again using trigonometric identities:

\[
S(t) = J_0(M) \sin (2\pi f_c t) \\
+ J_1(M) \sin [2\pi (f_c + f_m)] t - \sin [2\pi (f_c - f_m)] t \\
+ J_2(M) \sin [2\pi (f_c + 2f_m)] t + \sin [2\pi (f_c - 2f_m)] t \\
+ J_3(M) \sin [2\pi (f_c + 3f_m)] t - \sin [2\pi (f_c - 3f_m)] t
\]
2.2.2 Example

Now to illustrate spectral evolution let us take an example. Suppose \( f_c = 200\text{Hz}, f_m = 100\text{Hz} \) and \( f_d = 100\text{Hz} \) \((M=1)\). The Bessel Function values for \( M=1 \) are: \( J_0 = .77; J_1 = .44; J_2 = .11; J_3 = .02 \). This will produce the spectrum illustrated in Figure 2.5. If the amount of frequency deviation is increased from 100Hz to 300Hz \((M=3)\), the result shown in Figure 2.6 is obtained.

Since \( \sin(-x) = -\sin(x) \), the negative frequency components get reflected and add vectorially to the positive frequency components. One will notice, however, that the spectrum has "spread out" yielding higher frequency components of significant weight. Essentially, some of the energy from the carrier signal has been distributed to the other higher and lower frequency components. This, finally, is what is meant by spectrum evolution. As the spectrum continues to broaden, the timbre of the note gets richer and more interesting \([4]\).

One can control this dynamic action of the spectrum by simply applying a time envelope on the modulation index parameter. This technique is used in the FMMS. In addition, the FMMS has another time envelope which is applied to the amplitude of the output wave yielding this final FM equation:

\[
S(nt) = A(nt) \sin [2\pi f_c nt + M(nt) \sin (2\pi f_m nt)] \quad (10)
\]
Figure 2.5 FM SPECTRUM (M=1)

Figure 2.6 FM SPECTRUM (M=3)
Each instrument, therefore, will have three controlling parameters: \(A(nT)\), the output amplitude envelope function; \(M(nT)\), the modulation index envelope function and \((f_c/f_m)\), the ratio of the carrier frequency to the modulating frequency. If one represents this ratio in the form of \(N_1/N_2\), where \(N_1\) and \(N_2\) have all common factors divided out, then the fundamental frequency of the modulated carrier will be:

\[
f_o = f_c/N_1 = f_m/N_2
\]  \hspace{1cm} (11)

By making the ratio of \(f_c\) to \(f_m\) a small integer, a harmonic spectrum results (as can be seen in the example). However, if this ratio is non-integral, an inharmonic spectrum results, which should be useful in simulating percussive sounds.
CHAPTER 3

SYSTEM DESIGN

This and the next chapter deal with the hardware design of the FMMS. This chapter describes each subsystem in block diagram form and then puts the units together to give the reader a view of the overall system. Most of the specifics in the design are presented in chapter four.

In order to realize equation (10) as a digital hardware system, one needs to have a modulating frequency generator, a carrier frequency generator, a multiplier, an amplitude and modulation index envelope generators and a DAC.
3.1 Frequency Generators

The frequency generators are essentially phase angle generators. The idea is to load a register with a value (phase angle increment) which is proportional to the frequency one wants generated. This value gets periodically added into an accumulator register whose output is used to address a sine wave table. When the accumulator overflows, a new cycle of the sine wave begins. Thus, the number of times the accumulator overflows per second, will be the frequency of the output sine wave. Loading the increment register with a larger value will cause the accumulator to overflow more often and thus increase the frequency of the output wave. This scheme is illustrated in Figure 3.1.

The output frequency is dependent on the phase angle increment \( I \), the sampling rate \( S \), and the sine table length \( L \), by the following equation:

\[
f = I \times \frac{S}{L}
\]  

(12)

These three values \( I, S \) and \( L \) depend on how elaborate a system one wants. The considerations made in determining them can be found in the next chapter. In order to increase the utilization of the sine wave table, the ROM (Read Only Memory) address was multiplexed between the modulating phase angle and the modulated carrier phase angle.
Figure 3.1 FREQUENCY GENERATOR BLOCK DIAGRAM
3.2 Envelope Generators

The envelopes generated by the FMMS are concatenated linear segments whose slopes, which are preprogrammed, are loaded into a slope increment register. Like the frequency generators, this increment gets added periodically into an accumulator register. Unlike the frequency generators, however, where overflow starts a new cycle of the waveform, the envelope accumulator overflow simply increments or decrements a ramp register. Figure 3.2 illustrates this process. This design technique was used to shorten the register lengths which would need to be twice as large if the technique illustrated in Figure 3.1 were used.

Once again, the value loaded into the slope increment register depends on the sampling frequency and its value should be adjusted so that it never causes overflow in the ramp register. For the duration of the note, the CPU is responsible for updating the slope increment register to alter the slope of the ramp. \( M(nT) \) and \( A(nT) \) are generated in this part of the system.
Figure 3.2 ENVELOPE GENERATOR BLOCK DIAGRAM
3.3 Multiplier Circuit

In order to provide for the dynamic spectrum, the modulating frequency needs to be multiplied by a varying modulation index M(nT) (provided by the envelope generator subsystem). In addition, the modulated carrier wave is multiplied by A(nT) to provide for a dynamic output amplitude. These two functions were multiplexed into the multiplier input of a multiplier, and the output of the sine wave generator was used as the multiplicand. As one can begin to discern, the timing is very critical. Refer to the timing diagram in APPENDIX B for the complete details on the timing of the pipeline.

3.4 Digital-to Analog Conversion

Once each sample has been through the pipeline twice (once in forming the instantaneous modulation frequency and the second time forming the instantaneous modulated carrier frequency), it is sent to the output register which converts this digital data into an analog voltage. Then, after going through a low pass filter, it is sent out to the Music System amplifier and speaker. A 16-bit DAC would produce very low distortion, however it would cost hundreds of dollars. Therefore a 12-bit DAC was used, which for the FMMS is quite adequate.
Figure 3.3 shows a block diagram of the entire FMMS. In the next chapter, specifics concerning register length, sampling frequency, type of multiplier and sine wave table length are discussed.
Figure 3.3 FMMS BLOCK DIAGRAM
CHAPTER 4

HARDWARE DESIGN

The previous chapter illustrated the general hardware structure of the FMMS. This chapter describes the various considerations which were made in completing the design of the hardware.

4.1 Register Length

The first concern was how large to make the pipeline registers in the frequency generator and envelope generator circuits.
4.1.1 Frequency generator circuit

For the frequency generators, the register length depends on the maximum and minimum frequencies one would like to generate. Since the FMMS is to interface with the organ keyboard, the maximum allowable frequency should be $2^{13}-1$ Hz. An additional sign bit is needed since frequency modulation requires negative as well as positive frequencies.

To determine the minimum frequency, one must weigh cost as well as resolution. The most noticeable "frequency jitter" will be heard in the lower frequencies. Generally, comparisons are done around note C_2 = 65.4 Hz. The unit of measure is the cent (one cent = 1/1200 of an octave). Therefore:

$$1 \text{ cent} = 2^{1/1200} \quad (13)$$

To form a basis for comparison, some of the most elaborate systems have a frequency resolution of one cent at 65.4 Hz. Expensive tape recorders have wow and flutter figures around 0.1% or lower. At 65.4 Hz, this would be equivalent to 1.73 cents. In Table 4.1, cent deviation comparisons are made for registers having 14, 16 and 18 bits.
Table 4.1 ACCURACIES OF VARIOUS REGISTER LENGTHS

<table>
<thead>
<tr>
<th># of bits</th>
<th>f(min)</th>
<th>f(max)</th>
<th>cents dev. around 65.4Hz</th>
</tr>
</thead>
<tbody>
<tr>
<td>14</td>
<td>1.0000</td>
<td>8191</td>
<td>26.30</td>
</tr>
<tr>
<td>16</td>
<td>0.2500</td>
<td>8191</td>
<td>6.60</td>
</tr>
<tr>
<td>18</td>
<td>0.0625</td>
<td>8191</td>
<td>1.65</td>
</tr>
</tbody>
</table>

Obviously, with 18-bit registers, one would be able to produce the kind of accuracy available on some of the most expensive systems. However to do this would require an 8-bit microprocessor to do triple precision arithmetic. Microprocessors are relatively slow at performing data manipulation in music synthesis applications, so this slowdown would be more detrimental than would the loss of accuracy in using a smaller register length. For this reason the FMMS uses 16-bit pipeline registers in the frequency generator circuit.

4.1.2 Envelope generator circuit

The decision on how large to make the envelope generator registers depends on how much noise can be tolerated after the multiplier circuit. Figure 4.1 illustrates the model used in evaluating the SNR (Signal-to-Noise Ratio) for the signal coming out of the multiplier.
The errors are due to truncation of the frequency and amplitude signals. For this calculation, the two input signals are considered to be fractions, therefore the error will be equivalent to the weight of the least significant bit position. As an example, consider \( f \) & \( a \) to be 8 bits each (including sign), then:

\[
e_a + e_f = 2^{-7} + 2^{-7} = 2^{-6} = e_T
\]

\[
\text{SNR} = 10 \log \left( \frac{1}{e_T^2} \right) = 36.12 \text{dB}
\]  

(14)

where the variance of the signal is normalized to 1. Table 4.2 lists the SNR's for several amplitude and frequency register lengths.
Table 4.2 MULTIPLIER OUTPUT SNR FOR VARIOUS INPUT REGISTER LENGTHS

<table>
<thead>
<tr>
<th>f   (bits)</th>
<th>a   (bits)</th>
<th>SNR  (dB)</th>
</tr>
</thead>
<tbody>
<tr>
<td>12</td>
<td>8</td>
<td>41.62</td>
</tr>
<tr>
<td>12</td>
<td>12</td>
<td>60.21</td>
</tr>
<tr>
<td>12</td>
<td>16</td>
<td>65.70</td>
</tr>
<tr>
<td>16</td>
<td>8</td>
<td>42.11</td>
</tr>
<tr>
<td>16</td>
<td>12</td>
<td>65.70</td>
</tr>
<tr>
<td>16</td>
<td>12</td>
<td>65.70</td>
</tr>
<tr>
<td>12</td>
<td>INFINITE</td>
<td>66.23</td>
</tr>
<tr>
<td>16</td>
<td>INFINITE</td>
<td>90.31</td>
</tr>
</tbody>
</table>
The 8-bit register values appear rather poor. A SNR around 60dB would be much better. For this reason, 12-bit registers are used in the envelope generator circuit.

4.2 Pipeline Clock

The next concern was the frequency at which to sample the data. By having the pipeline clock be a power of 2, the frequency increment values would be equal to the actual frequency desired, but shifted right or left. Clock frequency values of $2^{14}$ or $2^{15}$ were possibilities. By making it much higher, one decreases the throughput of the pipeline. To be able to completely recreate the original waveform, the sampling frequency should be at least $2 \frac{1}{2}$ times the highest frequency component. This requirement is not met using a clock frequency of $2^{14}$Hz since the highest fundamental frequency is $2^{13}$Hz. Therefore, $f_{clk} = 2^{15}$Hz in the FMMS.

4.3 Sine Wave Table

A sine wave table stored in ROM is probably the easiest way to convert an instantaneous phase into the sine function. Again, a decision must be made as to how long the table should be and how many bits each word in the table should have.
In order to properly evaluate what size table should be used, reference was made to an article written by Richard Moore entitled "Table Lookup Noise for Sinusoidal Digital Oscillators" [5]. In the article, Moore compares the worst-case $SN_eR$ (signal-to-error noise ratio) of various size tables using three methods of addressing the table: truncation, rounding and interpolation.

A study conducted at the Center for Computer Research in Music and Acoustics at Stanford University concluded that a 4096-word x 12-bit/word table would produce very negligible distortion [6]. This table, however, contained 360 degrees of the sine wave and truncated the lower address bits. Moore's tables contain only 180 degrees of the sine wave and therefore use the most significant address bit to determine the polarity of the sine wave. Thus, the table size mentioned above would be equivalent to a 2048-word x 11-bit/word table which Moore lists as having a $SN_eR$ of 54.4dB. The ADSL has an abundance of 2K x 8 EPROM's (Erasable-Programmable Read Only Memory) (2716) so two of these chips were used to build a 2K x 15-bit sine wave table. If truncation were used, this would yield a $SN_eR$ of 54.4dB (according to the article, adding more than $b$ bits, where $2^b = \text{memory length}$, does not improve the $SN_eR$ since then practically all the noise is due to the finite table length). By adding just three more adder chips a round-off adder was inserted, thus improving the $SN_eR$ to 60.4dB.
4.4 Multiplier Circuit

The multiplier circuit, which is the bottleneck of the pipeline, has to be fast enough to perform six multiplications in one-quarter of a pipeline cycle if the FMMS is to be expanded to six channels. This requires each multiplication to be completed in 1.27 microseconds. A cheap, but slow serial multiplier that can perform 2's complement, 8x8 multiplication is the AMD25LS14. It is expandable and can perform a 16-x 12-bit multiply in less than 1.2 microseconds, and costs less than $10 per chip. This chip works very well in the FMMS, although it does require some additional hardware to perform synchronization and clocking. (Refer to Figure A10 for the multiplier synchronization schematic.)
CHAPTER 5

SYSTEM SOFTWARE

This chapter deals with the software in the FMMS. The major task of the software is to provide dynamic control of the envelope generators, and supervise the inputs from the ADSL Music System's keyboard. Two programs will be described. The first is a test program which allows the user to experiment with different sounds by allowing him to write the various FM parameters directly into RAM (Random Access Memory) through the use of a monitor program (refer to MUSMON monitor listing in APPENDIX D). The second description is for a program which interfaces the FMMS directly to the keyboard port. This program allows the user to play the FMMS in real time. The parameters for the particular instrument being played will be stored in ROM.
For each note that is played, five registers have to be loaded with values related to the type of sound being synthesized. These registers consist of the modulating phase angle increment (MPA), the carrier phase angle increment (CPA), the constant frequency deviation (FD), the frequency deviation envelope (FENV) and the amplitude envelope (AENV).

5.1.1.1 Scaling. The registers in the frequency generator circuit contain 16 bits (2 fraction bits, 13 integer bits and 1 sign bit). Therefore the value which is loaded into MPA and CPA will be the desired frequency scaled up by four. For instance, a frequency of 1024Hz will have a phase angle increment value of 4096. Keep in mind that the fundamental frequency of the modulated carrier will depend on the ratio of $f_c$ to $f_m$ (refer back to chapter two).

The registers in the envelope generating circuit contain only 12 bits (11 magnitude bits and 1 sign bit). Since the value loaded into the frequency deviation registers is the product of the modulating frequency and the modulation index, this value will need to be scaled down. The maximum positive slope increment is $2^{11}-1$. This value will therefore represent the maximum modulation frequency ($2^{13}-1$) times the
maximum modulation index (8), for a segment with maximum slope \[(\text{max amp change})/(\text{min seg duration}) = 1/(1/8) = 8\]. This product yields \(2^{19-64}\), therefore each value must be right-shifted by 8 places. The maximum possible value for the frequency deviation is \(8x(2^{13}-1)\) or \(2^{16}-8\). Therefore this value must be right-shifted by 5 places.

Since each envelope consists of four segments (representing the attack, decay, sustain and release of the note), eight parameter values will be needed for each envelope. Each segment has an increment value (which may be negative) along with a duration value. The CPU uses the duration value as a counter to signal when a new segment increment value should be loaded.

5.1.1.2 Timing. In order to properly scale each instantaneous frequency, the parameters must be loaded into their respective pipeline register during the correct clock cycle. To do this, the CPU samples the state of the pipeline clock and after the appropriate number of cycles have elapsed, loads the next parameter into its corresponding pipeline register.
5.1.2 Program description

The test program allows the user to experiment with different sounds by programming his own values for the parameters listed in Table 5.1, into the FMMS. (Refer to APPENDIX C for program listing and sample test data.)

5.2 Keyboard Interface Program

In order to play music in real time, an input port on the FMMS is connected to an output port from the ADSL Music System keyboard processor board. When the keyboard processor detects that a note has been pressed or released, it sends an interrupt to the FMMS processor. The FMMS processor then checks the code in the input port. If it is all zeros it jumps to the released key routine. If not, it splits the code into a lower and upper nibble. It then takes the lower nibble (which is the code for the pitch of the note) and jumps to a table in ROM containing all the parameters for that particular pitch in its highest octave. Then it takes the upper nibble (containing a code for the octave of the note) and divides the frequency dependent parameters in the table by a number which will bring the note into the proper octave. These modified parameter values are then stored in a buffer.
<table>
<thead>
<tr>
<th>PARAMETER</th>
<th>RANGE</th>
<th>DESCRIPTION</th>
</tr>
</thead>
<tbody>
<tr>
<td>FC</td>
<td>0-7FFFH</td>
<td>CARRIER PHASE ANGLE INCREMENT</td>
</tr>
<tr>
<td>FM</td>
<td>0-7FFFH</td>
<td>MODULATION PHASE ANGLE INCREMENT</td>
</tr>
<tr>
<td>FD</td>
<td>0-07FFH</td>
<td>FREQUENCY DEVIATION</td>
</tr>
<tr>
<td>FENV1-4</td>
<td>0-07FFH (POS)</td>
<td>FREQUENCY DEVIATION ENVELOPE</td>
</tr>
<tr>
<td></td>
<td>OFFF-0801H (NEG)</td>
<td>INCREMENT SEGMENTS 1-4</td>
</tr>
<tr>
<td>AENV1-4</td>
<td>0-07FFH (POS)</td>
<td>AMPLITUDE ENVELOPE</td>
</tr>
<tr>
<td></td>
<td>OFFF-0801H (NEG)</td>
<td>INCREMENT SEGMENTS 1-4</td>
</tr>
<tr>
<td>SD1-4</td>
<td>01-FF</td>
<td>ENVELOPE SEGMENT DURATION</td>
</tr>
</tbody>
</table>
After the decoding, scaling and storing have been completed, the processor sends the parameters to a routine similar to the Test Program discussed in the previous section. For the third envelope segment (sustain), the processor loads a zero increment into the amplitude and frequency deviation envelope registers. This will cause them to hold their ramp registers constant. The processor then waits for a released key interrupt. Upon receipt of the interrupt, it sends out the last segment increment value (release), to the envelope registers. After the duration of the release segment, the processor clears all pipeline registers and waits for another keypress.
6.1 Future Expansion

The present version of the FMMS is single-channel. However, it represents the basic building block upon which larger, more sophisticated systems can be constructed. In the following paragraphs, a few suggestions are given on how the FMMS may be expanded into a multivoice system.

In order to generate multiple, independent, output waveforms, the existing hardware will need to be time-multiplexed. The limiting factor as to how many additional channels may be added will be the speed of the multiplying pipeline segment. In order to perform the multiplexing, each of the increment registers will write their values into a RAM. Once the RAM’s have been loaded, a special counter circuit steps through...
the memory, reading both the increment and the accumulated values. These values are then sent to an adder and the result is written back into the RAM accumulator location. This concept is illustrated in Figure 6.1. The special counting circuit has logic which executes the reads and writes at the proper times as well as clocking the holding registers.

In addition, the output circuit would require a separate output section for each channel. This is shown in Figure 6.2 for a system having six channels. A six-bit shift register, clocked at six times the pipeline frequency could be used to trigger the sample and hold (S/H) circuits.

6.2 Closing Remarks

The frequency modulation technique used in the FMMS appears to have enough versatility to be able to simulate common musical instrument timbres. More research needs to be done, however, on how the spectrum of different instruments evolves before one can program the FMMS parameters to "accurately" simulate such tones. Strange, yet interesting sounds are easy to produce through experimentation with the Test Program. The FMMS gives an added dimension to music composition in general and particularly to the ADSL Music System.
Figure 6.1 MODIFIED INCREMENT REGISTER FOR HANDLING MULTIVOICE
Figure 6.2 MODIFIED OUTPUT SECTION FOR HANDLING MULTIVOICE
APPENDIX A

FMMS HARDWARE SCHEMATICS
**Component Layout for Frequency Generator Board**

**Diagram Description**

- Components are labeled with IC numbers (e.g., IC08, IC09) and additional identifiers (e.g., MPA, CPA).
- Connections are denoted with lines and labels (e.g., VCC, GND).
- A table is included with components, signals, and connectors.

**Decription of Acronyms**

- **MPA**: Modulator Phase Angle
- **CPA**: Carrier Phase Angle
- **CMRA**: Carrier Modulator Phase Angle
- **CMFA**: Carrier Modulator Frequency Angle

**Table**

<table>
<thead>
<tr>
<th>Signal</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>IC08-R</td>
</tr>
<tr>
<td>2</td>
<td>IC07-D</td>
</tr>
<tr>
<td>3</td>
<td>IC06-A</td>
</tr>
<tr>
<td>4</td>
<td>IC05-C</td>
</tr>
<tr>
<td>5</td>
<td>IC04-B</td>
</tr>
<tr>
<td>6</td>
<td>IC03-M</td>
</tr>
<tr>
<td>7</td>
<td>IC02-G</td>
</tr>
<tr>
<td>8</td>
<td>IC01-A</td>
</tr>
</tbody>
</table>

**Notes**

- **SYSTEM BUS**: Connects various components.
- **FREQ/MULT**: Frequency board to multiplier board.
- **GND**: Ground.
- **CLR**: Clear.
- **DB**: Data bus.
- **MCA**: Modulation Index Times One Wave Amplitude.

**University of Illinois Advanced Digital Systems Lab**

**Frequency Generator Board**

**Component Layout & Connector List**

<table>
<thead>
<tr>
<th>Date</th>
<th>CHK'D BY</th>
</tr>
</thead>
<tbody>
<tr>
<td>8-8-81</td>
<td>K. G. RAMIREZ</td>
</tr>
</tbody>
</table>

**No. 1 of 3**
Figure A2 MPA CIRCUIT
Figure A3 CMPA CIRCUIT
Figure A4: Component Layout for Envelope Generator Board

Component Layout for Envelope Generator Board

- Description of Acronyms:
  MNU - Multiplier
  ENV - Frequency Deviation
  AENV - Amplitude Envelope
  FD - Frequency Deviation

System Bus Connector
Figure A8 COMPONENT LAYOUT FOR MULTIPLIERS BOARD
Figure A9 MULTIPLIER SELECTOR CIRCUIT
Figure A10 MULTIPLIER SYNCHRONIZATION CIRCUIT

NOTE: ONE-SHOT SET UP FOR 100NS DELAY TO ALLOW MCNO INPUTS TO SETTLE

REFERENCES: [7]
Figure A7 Mulitplier & Output Circuit
Figure A12 COMPONENT LAYOUT FOR PROCESSOR BOARD
Figure A13 CLOCK GENERATOR CIRCUIT
Figure A14 CENTRAL PROCESSING UNIT
Figure A16 SERIAL I/O & KEYBOARD PORT

REFERENCES: [7], [11]

NOTES:
- ALL EDGE CONNECTOR CONNECTIONS DESIGNATED BY
- ALL RS232 FACE-UP CONNECTIONS DESIGNATED BY
- ALL INTERNAL CONNECTIONS DESIGNATED BY
- MEMORY ADDRESS: 3a80, 3a81
- KEYBOARD PORT ADDRESS: 2800

UNIVERSITY OF ILLINOIS
ADVANCED DIGITAL SYSTEMS LAB

PROCESSOR BOARD
SERIAL I/O & KEYBOARD PORT

DRAWN BY: J. M. ZAMORA
DATE: 3/4/81

REV. DESCRIPTION DATE MADE BY

No. 5/5
APPENDIX B

SYSTEM TIMING DIAGRAM
Figure B1 SYSTEM TIMING DIAGRAM
APPENDIX C

TEST PROGRAM SOFTWARE LISTING
**THIS IS A TEST PROGRAM TO PLAY A NOTE WITH**

**DYNAMIC CONTROL OF THE MODULATION INDEX AND**

**AMPLITUDE ENVELOPES USING THE ADSL FM MUSIC**

**SYNTHESIZER.**

---

**TYPE THE FOLLOWING DATA INTO RAM FOR TRUMPET-LIKE SOUNDS:**

**NOTE: FOR WORDS ENTER LOW BYTE IN LOWER ADDRESS!**

<table>
<thead>
<tr>
<th>LOCATION</th>
<th>DATA</th>
<th>DESCRIPTION</th>
</tr>
</thead>
<tbody>
<tr>
<td>3840,41</td>
<td>01EEH</td>
<td>FENV1</td>
</tr>
<tr>
<td>3842,43</td>
<td>0406H</td>
<td>FM</td>
</tr>
<tr>
<td>3844,45</td>
<td>0066H</td>
<td>FD</td>
</tr>
<tr>
<td>3846,47</td>
<td>03F6H</td>
<td>AENV1</td>
</tr>
<tr>
<td>3848,49</td>
<td>0400H</td>
<td>F0</td>
</tr>
<tr>
<td>384A</td>
<td>02H</td>
<td>SD1</td>
</tr>
<tr>
<td>384B</td>
<td>02H</td>
<td>SD2</td>
</tr>
<tr>
<td>384C</td>
<td>04H</td>
<td>SD3</td>
</tr>
<tr>
<td>384D</td>
<td>03H</td>
<td>SD4</td>
</tr>
<tr>
<td>38A0,A1</td>
<td>0000H</td>
<td>FENV2</td>
</tr>
<tr>
<td>38A2,A3</td>
<td>0000H</td>
<td>FENV3</td>
</tr>
<tr>
<td>38A4,A5</td>
<td>0F88H</td>
<td>FENV4</td>
</tr>
<tr>
<td>38B0,B1</td>
<td>0EC3H</td>
<td>AENV2</td>
</tr>
<tr>
<td>38B2,B3</td>
<td>0EC0H</td>
<td>AENV3</td>
</tr>
<tr>
<td>38B4,B5</td>
<td>0FD0H</td>
<td>AENV4</td>
</tr>
</tbody>
</table>
00F0  OPRTA EQU 0F9H ; OUTPUT PORT A
00FA  OPRTB EQU 0FAH ; OUTPUT PORT B
0020  SETCL EQU 20H ; COMMAND TO SET CLEAR PIN
00FB  PORTC EQU 0FBH ; OUTPUT PORT C
FF01  PORTA EQU OFF01H ; DB LOW BYTE
3800  DATA EQU 3800H ; RAM DATA STARTING ADDRESS
0020  RIM EQU 20H ; RIM OPCODE
3871  TEST EQU 3871H ; BEGINNING ADDRESS OF PROGRAM TEST VALUE
38AE  FENVBUF EQU 38A0H ; STARTING ADDRESS OF FENV BUFFER STORAGE
38B0  AENVBUF EQU 38B0H ; STARTING ADDRESS OF AENV BUFFER STORAGE
38BE  AENVPTR EQU 38BEH ; AENV BUFFER POINTER

0800  ORG 0800H
0801  F3  DI
0807  21A0 28  LXI H,FENVBUF
080D  22AE 38  SHLD FENVPTR
080E  21B0 38  LXI H,AENVBUF
0810  22BE 38  SHLD AENVPTR
0812  AF  XRA A
0814  D3FB  OUT PORTC ; CLEAR PIPELINE REG'S
0817  2A40 38  LHLD DATA+64
081A  2201FF  SHLD PORTA ; LOAD FENV1
081C  3E59  MOV A,H
081F  327038  STA TEST-1 ; DID PROGRAM GET THIS FAR?
0820  20  DB RIM
0821  F21A08  JP  LOOK ; WAIT FOR CLK1 TO BE HIGH
0824  3E21  MVIC A,21H
0826  D3FB  OUT PORTC ; CLOCK-IN FENV1
0828  3E69  MVIC A,69H
LOAD FM
WAIT FOR 1 1/2 PIPELINE CLOCK CYCLES

LOAD FD
CLOCK-IN FD

TEST IF PROGRAM GOT THIS FAR
0869  LK25 EQU  $1
0869   20  EQU  $1
0869   B7  EQU  $1
0869   FA908  EQU  $1
086E   2A4838  ORA  A
0871   2201FF  JM   LK25 ; WAIT FOR CLK1 LOW
0874   3E26  OUT  PORTC ; CLOCK-IN FC
0876   D3FB  MVI  A,29H
0878   3E69  MVI  PORTA ; LOAD FC
087A   327438  STA  A,69H ; TEST IF PROGRAM GOT THIS FAR
087D   3A4A38  LDA  DATA+74 ; LOAD SD1
0880   4F  MOV  C,A
0881   CDCC08  CALL  WAIT
0884   OD  DCR  C
0885   C28108  JNZ  REP
0888   CD608  CALL  SEG
088B   3E69  MVI  A,69H ; TEST IF PROGRAM GOT THIS FAR
088D   327538  STA  TEST+3 ; TEST IF PROGRAM GOT THIS FAR
0890   3A4B38  LDA  DATA+75 ; LOAD SD2
0893   4F  MOV  C,A
0894   CDCC08  CALL  WAIT
0897   OD  DCR  C
0898   C29408  JNZ  REP05
089B   CD608  CALL  SEG
089E   3A4C38  LDA  DATA+76 ; LOAD SD3
08A1   4F  MOV  C,A
08A2   CDCC08  CALL  WAIT
08A5   OD  DCR  C
08A6   C2A208  JNZ  REP10
08A9   CD608  CALL  SEG
08AC 3A4D38
08AF 4F
08B0 CDCC08 REP15:
08B3 OD
08B4 C2B008
08B7 AF  OUT PORTC ; CLEAR PIPELINE REG'S
08B8 D3FB
08BA CDCC08 CALL WAIT
08BD CDCC08 CALL WAIT
08C0 CDCC08 CALL WAIT
08C3 CDCC08 CALL WAIT
08C6 CDCC08 CALL WAIT;
08C9 C30008 JMP 0800H ; REPEAT TONE

08CC 213620 WAIT: LXI H,11574D; DELAY FOR 1/8 SECOND
08CF 2B LOOP: DCX H
08D0 7C MOV A,H
08D1 B7 ORA A
08D2 C2CF08 JNZ LOOP
08D5 C9 RET

08D6 2AAE38 SEG: LHLD FENVPTR; LOAD NEXT FREQ. DEV.
08D9 7E MOV A,M
08DA D3F9 OUT OPRTA
08DC 23 INX H
08DD 7E MOV A,M
08DE D3FA OUT OPRTB
08E0 23 INX H
08E1 22AE38 SHLD FENVPTR
08E4 3E20 MVI A,SETCL
08E6 D3FB OUT PORTC ; LOWER CLOCK (IF SET)
08E8 3E21 MVI A,21H
08EA D3FB OUT PORTC ; CLOCK-IN FENV SEG
NO PROGRAM ERRORS

08EC 2ABE38  LHLDAENVPTR; LOAD NEXT AMP. ENVELOPE SEGMENT INCREMENT
08EF  7E     MOV A,M
08F0  D3F9   OUT OPRTA
08F2  23     INX H
08F3  7E     MOV A,M
08F4  D3FA   OUT OPRTB
08F6  23     INX H
08F7  22BE38 SHLDAENVPTR
08FA  3E30   MVI A,30H
08FC  D3FB   OUT PORTC; CLOCK-IN AENV REG

08FE  C9     RET
            END
# SYMBOL TABLE

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>0007</td>
</tr>
<tr>
<td>B</td>
<td>0000</td>
</tr>
<tr>
<td>C</td>
<td>0001</td>
</tr>
<tr>
<td>D</td>
<td>0002</td>
</tr>
<tr>
<td>E</td>
<td>0003</td>
</tr>
<tr>
<td>FENVB</td>
<td>38A0</td>
</tr>
<tr>
<td>FENVP</td>
<td>38AE</td>
</tr>
<tr>
<td>H</td>
<td>0004</td>
</tr>
<tr>
<td>L</td>
<td>0005</td>
</tr>
<tr>
<td>LK05</td>
<td>082D</td>
</tr>
<tr>
<td>LK10</td>
<td>0838</td>
</tr>
<tr>
<td>LK15</td>
<td>083D</td>
</tr>
<tr>
<td>LK20</td>
<td>0847</td>
</tr>
<tr>
<td>LK25</td>
<td>0859</td>
</tr>
<tr>
<td>LK25</td>
<td>0859</td>
</tr>
<tr>
<td>LOOP</td>
<td>086A</td>
</tr>
<tr>
<td>M</td>
<td>0006</td>
</tr>
<tr>
<td>OPRTA</td>
<td>00F9</td>
</tr>
<tr>
<td>OPRTB</td>
<td>00FA</td>
</tr>
<tr>
<td>OPRT15</td>
<td>08B0</td>
</tr>
<tr>
<td>OPRT15</td>
<td>08B0</td>
</tr>
<tr>
<td>PORTA</td>
<td>FF01</td>
</tr>
<tr>
<td>PORTB</td>
<td>00FA</td>
</tr>
<tr>
<td>PORTC</td>
<td>00FB</td>
</tr>
<tr>
<td>PSW</td>
<td>0006</td>
</tr>
<tr>
<td>REPO5</td>
<td>0894</td>
</tr>
<tr>
<td>SEG</td>
<td>08D6</td>
</tr>
<tr>
<td>SETCL</td>
<td>0020</td>
</tr>
<tr>
<td>SP</td>
<td>0006</td>
</tr>
<tr>
<td>TEST</td>
<td>3871</td>
</tr>
<tr>
<td>WAIT</td>
<td>08CC</td>
</tr>
<tr>
<td>DATA</td>
<td>3800</td>
</tr>
<tr>
<td>DE</td>
<td>0809</td>
</tr>
<tr>
<td>H</td>
<td>0004</td>
</tr>
<tr>
<td>LK15</td>
<td>083D</td>
</tr>
<tr>
<td>LK20</td>
<td>0847</td>
</tr>
<tr>
<td>LK25</td>
<td>0859</td>
</tr>
<tr>
<td>LOOP15</td>
<td>086A</td>
</tr>
<tr>
<td>M</td>
<td>0006</td>
</tr>
<tr>
<td>OPRT15</td>
<td>08B0</td>
</tr>
<tr>
<td>PORTA</td>
<td>FF01</td>
</tr>
<tr>
<td>PORTB</td>
<td>00FA</td>
</tr>
<tr>
<td>PORTC</td>
<td>00FB</td>
</tr>
<tr>
<td>PSW</td>
<td>0006</td>
</tr>
<tr>
<td>REPO5</td>
<td>0894</td>
</tr>
<tr>
<td>SEG</td>
<td>08D6</td>
</tr>
<tr>
<td>SETCL</td>
<td>0020</td>
</tr>
</tbody>
</table>
APPENDIX D

FMMS MUSMON MONITOR LISTING
; LAST MODIFICATION 4:45 PM 7/25/81
; BY BOB KAMINSKY

0000 KBPROC EQU 0 ; KEYBOARD PROC. INPUT PORT
0008 KBRST EQU 6 ; OUTPUT PORT TO RESET KBD PROC
38C0 RSTORG EQU 38COH ; WHERE RESTART JUMP TABLE IS
0800 KEYPROC EQU 0800H ; 2ND ROM SLOT LOCATION
0010 DELFAC EQU 10H ; DEFAULT DELAY FACTOR FOR MONITOR
0030 SIM EQU 30H ; SIM OPCODE
0020 RIM EQU 20H ; RIM OPCODE

4021 BAUD EQU 33+4000H ; FOR 4800 BAUD SQUARE WAVE
38F0 IOTBL EQU 38FOH ; I/O TABLE
001B BRCHR EQU 1BH
383D BRLOC EQU 383DH ; BREAK LOCATION
0027 CMD EQU 27H ; USART COMMAND WORD
0031 CNCTL EQU 031H ; CONSOLE CONTROL
0030 CNIN EQU 030H ; CONSOLE IN
0030 CNOUT EQU 030H ; CONSOLE OUT
0031 CONST EQU 031H
00F8 CTCTL EQU 0F8H ; COUNTER/TIMER CONTROL
00CF CTCMD EQU 0CFH ; COUNTER/TIMER COMMAND
000D CR EQU 00H
3800 DATA EQU 3800H
001B ESC EQU 1BH
000F HCHAR EQU 0FH
00FF INVRT EQU 0FFH
000A LF EQU 0AH
00CE MODE EQU 0CEH
0007 NEWLN EQU 07H
007F PRTYO EQU 7FH
382E REGS EQU DATA+64-18
0002 RBR EQU 2
0038 RSTU EQU 38H
001B TERM EQU 1BH
0001 TRDY EQU 1
00FF UPPER EQU 0FFH
0004 TXBE EQU 04H
00FC TLO EQU 0FCH ; LOW ORDER 6 DIVISION
00FD THI EQU 0FDH ; LOWER 6 BITS=HIGH DIVISION RATE
COLD START SYSTEM

<table>
<thead>
<tr>
<th>Memory Address</th>
<th>Opcode</th>
<th>Instruction</th>
</tr>
</thead>
<tbody>
<tr>
<td>0000h</td>
<td>F3</td>
<td>DJ</td>
</tr>
<tr>
<td>0001h</td>
<td>C35100</td>
<td>JMP STUP</td>
</tr>
<tr>
<td>0008h</td>
<td>C3C038</td>
<td>JMP RS1</td>
</tr>
<tr>
<td>0010h</td>
<td>C3C338</td>
<td>JMP RS2</td>
</tr>
<tr>
<td>0018h</td>
<td>C3C638</td>
<td>JMP RS3</td>
</tr>
<tr>
<td>0020h</td>
<td>C3C938</td>
<td>JMP RS4</td>
</tr>
<tr>
<td>0024h</td>
<td>C3DE38</td>
<td>JMP TRAP</td>
</tr>
<tr>
<td>0028h</td>
<td>C3CC38</td>
<td>JMP RS5</td>
</tr>
<tr>
<td>002Ch</td>
<td>C3CF38</td>
<td>JMP RS65</td>
</tr>
<tr>
<td>0030h</td>
<td>C3D238</td>
<td>JMP RS6</td>
</tr>
<tr>
<td>0034h</td>
<td>C3D538</td>
<td>JMP RS65</td>
</tr>
<tr>
<td>0038h</td>
<td>C3D838</td>
<td>JMP RS7</td>
</tr>
<tr>
<td>003Ch</td>
<td>C3DB38</td>
<td>JMP RS75</td>
</tr>
<tr>
<td>003Fh</td>
<td>C3E00</td>
<td>JMP WARM</td>
</tr>
<tr>
<td>0042h</td>
<td>C39302</td>
<td>JMP CHAR</td>
</tr>
<tr>
<td>0045h</td>
<td>C3D602</td>
<td>JMP GETCH</td>
</tr>
<tr>
<td>0048h</td>
<td>C38703</td>
<td>JMP NMOUT</td>
</tr>
<tr>
<td>004Bh</td>
<td>C3DD02</td>
<td>JMP GETHX</td>
</tr>
<tr>
<td>004 Eh</td>
<td>C3E02</td>
<td>JMP PRMSG</td>
</tr>
</tbody>
</table>

JUMP TABLE FOR ACCESSING MONITOR ROUTINES

<table>
<thead>
<tr>
<th>Memory Address</th>
<th>Opcode</th>
<th>Instruction</th>
</tr>
</thead>
<tbody>
<tr>
<td>003Fh</td>
<td>C3E00</td>
<td>JMP WARM</td>
</tr>
<tr>
<td>0042h</td>
<td>C39302</td>
<td>JMP CHAR</td>
</tr>
<tr>
<td>0045h</td>
<td>C3D602</td>
<td>JMP GETCH</td>
</tr>
<tr>
<td>0048h</td>
<td>C38703</td>
<td>JMP NMOUT</td>
</tr>
<tr>
<td>004Bh</td>
<td>C3DD02</td>
<td>JMP GETHX</td>
</tr>
<tr>
<td>004 Eh</td>
<td>C3E02</td>
<td>JMP PRMSG</td>
</tr>
</tbody>
</table>
STUP: MVI A,37H ; DUMMY CMD WD TO UART TO SET UP SOFT RESET
OUT CNCTL

; SET UP RESTART JUMP TABLE IN RAM
; TO GO TO THE ROM BREAKPOINT
; ROUTINE
MVI A,40H ; SOFTWARE RESET USART
; OUTPUT MODE TO USART
MVI A,MODE OUT CNCTL

MVI D,10 ; COUNTER
LXI H,RSTORG ; POINT TO RAM RST TABLE
LXI B,GO
STLP: MVI M,0C3H ; JUMP OPCODE
INX H
MOV M,C ; ADDRESS OF "GO"
INX H
MOV M,B
INX H
DCR D ; CHECK COUNTER
JNZ STLP ; REPEAT LOOP
MVI A,11011111B ; DISABLE INTERRUPT LEVELS
DB SIM ; SIM INSTRUCTION
JMP INUST ; BRANCH TO COMPLETE INIT

SHLD LSAVE
POP H
SHLD PSAVE
PUSH PSW
LXI H,2
DAD SP
SHLD SSAVE
POP PSW
LXI SP,ASAVE+1
JMP ADROUT
;******************************************************************************
; PRINT SIGNON MESSAGE SYSTEM
;******************************************************************************

WARM: ; WARM START ENTRY POINT
SOMSG: LXI H,SIGNON
        CALL PRMSG
        JMP GETCM

;******************************************************************************
; COMMAND RECOGNIZING ROUTINE
;******************************************************************************

GETCM: LXI SP,MSTAK
        MVI C, '>
        CALL ECHO
GTC03:   CALL GETCH
GTC05:   CMP M
            JZ GTC10
            INX H
            OCR C
            JNZ GTC05
            JMP ERROR
GTC10:   LXI H,CADR
            DAD B
            DAD B
            MOV A,M
            INX H
            MOV H,M
            MOV L,A
            DCMD: MVI C, 2

DCMD:   MVI C, 2
        POP D
        POP H
        CALL CROUT
        CALL ADRD
00CD 0E20 OCM10: MVI C, ' '  
00CF CDAP02 CALL ECHO  
00D2 007E MOV A, M  
00D3 CD0733 CALL NMOUT  
00D6 C06102 CALL BREAK  
00D9 DACC02 JC EXIT  
00DC CD5603 CALL HILO  
00DF DACD02 JC EXIT  
00E2 0230 INX H  
00E3 027D MOV A, L  
00E4 060F ANI OFFH  
00E6 C2C600 JNZ DCM10  
00E9 C3C700 JMP DCM05  

00EC DDD0D2 GCMD: CALL GETHX  
00EF D20101 JNC GCM05  
00F2 07A MOV A, D  
00F3 FE0D CPI CR  
00F5 C2C802 JNZ ERROR  
00F6 213638 LXI H, PSAVE  
00FB 071 MOV M, C  
00FC 023 INX H  
00FD 070 MOV M, B  
00FE C30701 JMP GCM10  
0101 07A GCHO5: MOV A, D  
0102 FE0D CPI CR  
0104 C2C802 JNZ ERROR  
0107 C3EC03 GCM10: JMP RSTTF  

010A 21CC05 GCMD: LXI H, CMSG  
010D CD9E02 CALL PRMSG  
0110 C39500 JMP GETCM
NEW FORMAT FOR OUTPUT COMMAND:
0 PORT, <DATA>

OCMD: CALL GETHX

OC05: PUSH B ; PUT PORT # ON STACK
CALL GETHX
JNC ERROR
MOV A,D
FE20 CPI
CA2401 JZ OC05
FE2C CPI
C2802 JNZ ERROR
C55

OC05: PUSH B ; PUT DATA ON STACK
CALL SETUP
MOV A,C
D1 POP D
7B MOV A,E
32F138 STA IOTBL+1
79 MOV A,C
CDF038 CALL IOTBL
C9500 JMP GETCM

OCMD: CALL GETHX
D2802 JNC ERROR
7A MOV A,D
FEOD CPI CR
C2802 JNZ ERROR
C55

ICMD: CALL GETHX
D2802 JNC ERROR
7A MOV A,D
FE3D CPI CR
CA5301 JZ IC01
FE0D CPI CR
C2802 JNZ ERROR
C55

IC01: PUSH B
CD3902 CALL SETUP
C1 POP B
79 MOV A,C
32F438 STA IOTBL+4
CDF338 CALL IOTBL+3
4F MOV C,A
CD8703 CALL NMOUT
CDAA02 CALL CROUT
C39500 JMP GETCM
0169 OE03 MCMD: MVI C,3
016B CD1103 CALL GETNM
016E C1 POP B
016F E1 POP H
0170 D1 POP D
0171 CD7701 CALL MCM05
0174 C39500 JMP GETCM
0177 E5 MCM05: PUSH H
0178 62 MOV H,D
0179 6B MOV L,E
017A 7E MOV A,M
017B 60 MOV H,B
017C 69 MOV L,C
017D 77 MOV M,A
017E 03 INX B
017F 7E MOV A,B
0180 B1 ORA C
0181 C8 R2
0182 13 INX D
0183 E1 POP H
0184 CD5603 CALL HILO
0187 D0 RNC
0188 C37701 JMP MCM05
018B OE01 SCMD: MVI C,1
018D CD1103 CALL GETNM
0190 E1 POP H
0191 C39F01 JMP SCM10
0194 7A SCM05: MOV A,D
0195 FE20 CPI 
0197 CA9F01 JZ SCM10
019A FE2C CPI 
019C C29500 JNZ GETCM
019E 7E MOV A,M
019F CD8703 CALL NMOUT
01A0 OE2D MVI C,'-
01A5 CDAF02 CALL ECHO
01A8 CDDD02 CALL GETHX
01AB D2AF01 JNC SCM15
01AE 71 MOV M,C
01AF 23 SCM15: INX H
01B0 7D MOV A,L
01B1 E60F ANI OFH
01B3 C29401 JNZ SCM05
01B6 CDAA02 CALL CROUT
01B9 CD4702 CALL ADRD
01BC 0E20 MVI C,
01BE CDAF02 CALL ECHO
01C1 C39401 JMP SCM05
01C4 CDD602 XCMD: CALL GETCH
01C7 CDAF02 CALL ECHO
01CA 79 MOV A,C
01CB FE0D CPI CR
01CD C2D601 JNZ XCM05
01DD CDA403 CALL REGDS
01D3 C39500 JMP GETCM
01D6 4F XCM05: MOV C,A
01D7 CDD503 CALL RGADR
01DA 05 PUSH B
01DB EI POP H
01DC 0E20 MVI C,
01DE CDAF02 CALL ECHO
01E1 79 MOV A,C
01E2 323A38 STA TEMP
01E5 3A3A38 XCM10: LDA TEMP
01E8 FE20 CPI
01EA CAF201 JZ XCM15
01ED FE2C CPI
01EF C29500 JNZ GETCM
01F2 7E XCM15: MOV A,M

01F3 B7 ORA A
01F4 CACD02 JZ EXIT
01F5 E5 PUSH H
01F6 8E MOV E,M
01F7 1638 MVI D,DATA/256
01F8 23 INX H
01FC 46 MOV B,M
01FD D5 PUSH D
01FE D5 PUSH D
01FF E1 POP H
0200 C5 PUSH B
0201 7E MOV A,M
0202 CD8703 CALL NMOUT
0205 F1 POP PSW
0206 F5 PUSH PSW
0207 B7 ORA A
0208 CA1002 JZ XCM20
020B 2B DCX H
020C 7E MOV A,M
020D CD8703 CALL NMOUT
0210 0E2D XCM20: MVI C,'-'"
0212 CDAF02 CALL ECHO
0215 CDDD02 CALL GETHX
0218 D23002 JNC XCM30
021B 7A MOV A,D
021C 323A35 STA TEMP
021F F1 POP PSW
0220 E1 POP H
0221 87 ORA A
0222 CA2702 JZ XCM25
0225 70 MOV M,B
0226 2B DCX H
XCM25: MOV M, C
XCM27: LXI D, RTABS
XCM30: MOV A, D

SETUP: LXI B, IOTBL
LXI D, TBL1
LXI H, 5
DAD D
CALL MCM05
RET

UTILITY

INPUTS: HL-ADDRESS TO BE DISPLAYED
CALLS: NMOUT
DESTROYS: A
FUNCTION: OUTPUTS TO THE CONSOLE THE ADDRESS CONTAINED IN THE HL REGISTER PAIR

ADRD: MOV A, H
CALL NMOUT
MOV A, L
CALL NMOUT
RET

ADROUT: PUSH PSW
PUSH B
PUSH D
MVI C, '#'
CALL ECHO
LHLD PSAVE
CALL ADRD
JMP EXIT
**BREAK**

Outputs: Carry=1 if escape character input
0 if any other character or no character

Destroys: A,F/F's

Function: Break is used to sense an escape character from the user. If no character is pending, or if the character is not an escape, then a failure return (carry=0) is taken. In this case, the character is lost. If the character is an escape, then return is successful (carry=1).

<table>
<thead>
<tr>
<th>Address</th>
<th>Opcode</th>
<th>Instruction</th>
</tr>
</thead>
<tbody>
<tr>
<td>0261</td>
<td>DB31</td>
<td>Break: In Const ; get UART status</td>
</tr>
<tr>
<td>0263</td>
<td>E602</td>
<td>ANI RBR</td>
</tr>
<tr>
<td>0265</td>
<td>CAD302</td>
<td>JZ FRET ; return failure if no char</td>
</tr>
<tr>
<td>0268</td>
<td>DB30</td>
<td>BR05: In CNIN ; else, get char</td>
</tr>
<tr>
<td>026A</td>
<td>E67F</td>
<td>ANI PRTY0</td>
</tr>
<tr>
<td>026C</td>
<td>FE1B</td>
<td>CPI BCHR</td>
</tr>
<tr>
<td>026E</td>
<td>CA0004</td>
<td>JZ SRET ; if escape, then successful return</td>
</tr>
<tr>
<td>0271</td>
<td>FE13</td>
<td>CPI 13H</td>
</tr>
<tr>
<td>0273</td>
<td>C2D302</td>
<td>JNZ FRET ; if not &lt;CTR&gt;S, then fail return</td>
</tr>
<tr>
<td>0276</td>
<td>DB31</td>
<td>BR10: In Const ; else, wait for new char</td>
</tr>
<tr>
<td>0278</td>
<td>E602</td>
<td>ANI RBR</td>
</tr>
<tr>
<td>027A</td>
<td>CA7602</td>
<td>JZ BR10</td>
</tr>
<tr>
<td>027D</td>
<td>C36802</td>
<td>JMP BR05</td>
</tr>
</tbody>
</table>

**Cl**

Outputs: A-character from console

Destroys: A,F/F's

Function: Cl waits until a character has been entered at the console and then returns the character, via the A register, to the calling routine.

<table>
<thead>
<tr>
<th>Address</th>
<th>Opcode</th>
<th>Instruction</th>
</tr>
</thead>
<tbody>
<tr>
<td>0280</td>
<td>DB31</td>
<td>Cl: In Const</td>
</tr>
<tr>
<td>0282</td>
<td>E602</td>
<td>ANI RBR</td>
</tr>
<tr>
<td>0284</td>
<td>CA8002</td>
<td>JZ Cl</td>
</tr>
<tr>
<td>0287</td>
<td>DB30</td>
<td>IN CNIN</td>
</tr>
<tr>
<td>0289</td>
<td>C9</td>
<td>RET</td>
</tr>
</tbody>
</table>
INPUTS: C-ASCII CHARACTER '0'-'9' OR 'A'-'F'

OUTPUTS: A-0 7F F HEX

DSTROYS: A,F/F'S

FUNCTION: CNVBN CONVERTS THE ASCII REPRESENTATION OF A HEX CHARACTER INTO ITS CORRESPONDING BINARY VALUE. CNVBN DOES NOT CHECK THE VALIDITY OF ITS INPUT.

CNVBN: MOV A,C
      SUI '0'
      CPI 10D
      RM
      SUI 7
      RET

CO: IN CONST
    ANI TROY
    JZ co
    MOV A,C
    OUT CNOUT
    RET

PRMSG: MOV A,M
        CPI OFFH ; END OF STRING?
        RZ
        MOV C,M
        CALL co
        INX H
        JMP PRMSG
CROUT

UTILITY

CALLS: ECHO

DESTROYS: A,B,C,F/F'S

FUNCTION: CROUT SENDS A CARRIAGE RETURN (AND HENCE A LINE FEED) TO THE CONSOLE DEVICE.

THE CONSOLE DEVICE.

MVI C, CR
JMP ECHO

ECHO

UTILITY

INPUTS: C-CHARACTER TO ECHO TO CONSOLE

OUTPUTS: C-CHARACTER ECHOED TO CONSOLE

CALLS: CO

DESTROYS: A,B,F/F'S

FUNCTION: ECHO TAKES A SINGLE CHARACTER AS INPUT AND, VIA THE MONITOR, SENDS THAT CHARACTER TO THE USER CONSOLE. A CARRIAGE RETURN IS ECHOED AS A CARRIAGE RETURN LINE FEED, AND AN ESCAPE IS ECHOED AS ' $ '.

MOV B, C
MVI A, ESC
CMP B
JNZ ECHO5
MVI C, '$'
ECH05: CALL CO
MVI A, CR
CMP B
JNZ ECH10
MVI C, LF
CALL CO
ECH10: MOV C, B
RET
;******************************************************************************
ERROR
CALLS: ECHO, CROUT, GETCM
DESTROYS: A, B, C, F/F'S
FUNCTION: ERROR PRINTS THE AN ASTERISK ON THE CONSOLE, FOLLOWED BY
A CARRIAGE RETURN LINE FEED, AND THEN RETURNS CONTROL
TO THE COMMAND RECOGNIZER.
;******************************************************************************
ERROR: MVI C, 'C'
CALL ECHO
EXIT: CALL CROUT
JMP GETCM
;******************************************************************************
FRET
CALLS: CROUT
DESTROYS: CARRY
FUNCTION: FRET SETS THE CARRY FALSE, DENOTING FAILURE, AND THEN
RETURNS TO THE CALLER OF THE ROUTINE INVOKING FRET.
;******************************************************************************
FRET: STC
CMC
RET
;******************************************************************************
GETCH
DESTROYS: C-NEXT CHARACTER IN INPUT STREAM
CALLS: C1
DESTROYS: A, C, F/F'S
FUNCTION: GETCH RETURNS THE NEXT CHARACTER IN THE INPUT STREAM TO
THE CALLING ROUTINE.
;******************************************************************************
GETCH: CALL C1
ANI PRTY0
MOV C, A
RET
**GETHEX**

**OUTPUTS:** BC-16 BIT INTEGER

**D-CHARACTER WHICH TERMINATED INTEGER**

**CARRY=1 IF FIRST CHARACTER NOT TERMINATOR**

**=0 IF FIRST CHARACTER IS TERMINATOR**

**CALLS:** GETCH, ECHO, VALDL, VALDG, CVNBN, ERROR

**DESTROYS:** A, B, C, D, E, F/F'S

**FUNCTION:** GETHX ACCEPTS A STRING OF HEX DIGITS FROM THE INPUT STREAM AND RETURNS THEIR VALUE AS A 16-BIT BINARY INTEGER. IF MORE THAN 4 HEX DIGITS ARE ENTERED, ONLY THE LAST 4 ARE USED. THE NUMBER TERMINATES WHEN A VALID DELIMITER IS ENCOUNTERED. THE TERMINATOR IS RETURNED AS AN OUTPUT OF THE FUNCTION. ILLEGAL CHARACTERS CAUSE AN ERROR INDICATION. IF THE FIRST CHARACTER IN THE INPUT STREAM IS NOT A VALID DELIMITER, GETHX WILL RETURN WITH THE CARRY BIT SET TO 1; OTHERWISE, THE CARRY BIT IS SET TO 0 AND THE CONTENTS OF BC PAIR ARE UNDEFINED.

```assembly
02DD E5 GETHX: PUSH H
02DE 210000 LXI H, 0
02E0 1E00 MVI E, 0
02E3 CDD602 GHX05: CALL GETCH
02E6 CD4F02 CALL ECHO
02E9 CD4704 CALL VALDL
02EC D2FB02 JNC GHX10
02EF 51 MOV D, C
02F0 E5 PUSH H
02F1 C1 POP B
02F2 E1 POP H
02F3 7B MOV A, E
02F4 B7 ORA A
02F5 C20004 JNZ SRET
02F8 CAD302 JZ FRET
02FB CD2C04 GHX10: CALL VALDG
02FE D2C802 JNC ERROR
0301 CD8A02 CALL CVNBN
0304 1EFF MVI E, INVRT
0306 29 DAD H
0307 29 DAD H
0308 29 DAD H
```
GETNM

UTILITY

INPUTS: C-COUNT OF NUMBERS TO FIND IN INPUT STREAM
OUTPUTS: TOP OF STACK-NUMBERS FOUND IN REVERSE ORDER
CALLS: GETHX, HILO, ERROR
DESTROY: A, B, C, D, E, H, L, F/F'S
FUNCTION: GETNM FINDS A SPECIFIED COUNT OF NUMBERS, BETWEEN 1 AND 3,
          INCLUSIVE, IN THE INPUT STREAM AND RETURNS THEIR VALUES
          ON THE STACK. IF 2 OR MORE NUMBERS ARE REQUESTED, THEN
          THE FIRST MUST BE LESS THAN OR EQUAL TO THE SECOND, OR
          THE FIRST AND SECOND NUMBERS WILL BE SET EQUAL. THE LAST
          NUMBER REQUESTED MUST BE TERMINATED BY A CARRIAGE RETURN
          OR AN ERROR WILL RESULT.

******************************************************************************

GETNM:

MVI L,03
MOV A,C
ANI 03
RZ
MOV H,A

GNM05:
CALL GETHX
JNC ERROR
PUSH B
OCR L
OCR H
JZ GNM10
MOV A,D
CPI CR
JZ ERROR
JMP GNM05

GNM10:
MOV A,D
CPI CR
JNZ ERROR
LXI B,OFFFH
MOV A,L
ORA A
JZ GNM20

******************************************************************************
**HIL0: PUSH B**

**MVI B,A**

**INX H**

**MOV A,H**

**ORA L**

**DCX H**

**STC**

**JZ HLO5**

**MOV A,L**

**SUB E**

**MOV A,H**

**SBB D**

---

**HIL0 UTILITY**

**INPUTS:** DE-16 BIT INTEGER

**HL-16 BIT INTEGER**

**OUTPUTS:** CARRY=0 IF HL<DE

=1 IF HL>=DE

**DESTROYS:** A,F/F'S

**FUNCTION:** HIL0 COMPARES THE TWO 16-BIT INTEGERS IN HL AND DE PAIRS.

THE INTEGERS ARE TREATED AS UNSIGNED TWO'S COMPLEMENT NUMBERS. THE CARRY BIT IS SET ACCORDING TO THE RESULT OF THE COMPARISON.
NMOUT: PUSH RRC RRC RRC RRC CALL CALL CALL CALL CALL RET PSW PSW PSW PRVAL ECHO PDP PSW PRVAL ECHO ECHO
**PRVAL**

**UTILITY**

**INPUTS:** A-INTEGER, RANGE 0 TO F

**OUTPUTS:** A-ASCII CHARACTER

**DESTROYS:** C

**FUNCTION:** PRVAL CONVERTS THE NUMBER IN THE A REGISTER FROM 0 TO F INTO THE CORRESPONDING ASCII CHARACTER, 0-9, A-F. PRVAL DOES NOT CHECK THE VALIDITY OF THE INPUT.

```
PRVAL: ANI HCHAR
ADI 90H
DAA
ACI 40H
DAA
MOV C, A
RET

REGDS: LXI H, RTAB
REG05: MOV C, M
MOV A, C
JNZ REG10
CALL CROUT
REG10: CALL ECHO
MVI c, I = I
CALL ECHO
INX H
MOV E, M
MVI D, DATA/256
INX H
LDAX D
CALL NMOUT
MOV A, M
JZ REG15
DCX D
LDAX D
CALL NMOUT
REG15: MVI c, 'A'
CALL ECHO
INX H
JMP REG05
```
SRET: STC

SRET UTILITY

OUTPUTS: CARRY=1

DESTROYS: CARRY

FUNCTION: SRET IS CALLED BY ROUTINS WISHING TO RETURN SUCCESS.
SRET SETS THE CARRY TRUE AND THEN RETURNS TO THE CALLER OF THE ROUTINE INVOKING SRET.
0402 3A3A38  STH0:  LDA  TEMP
0405 B7    ORA  A
0406 CO    RNZ  
0407 0E00  MVI  C,0
0409 CD0D04 CALL STHLF
040C C9    RET

040D D5    STHLF: PUSH D
040E E1    POP  H
040F 79    MOV  A,C
0410 E60F  ANI  HCHAR
0412 4F    MOV  C,A
0413 3A3A38 LDA  TEMP
0415 B7    ORA  A
0417 C2204 CALL STH05
041A 7E    MOV  A,M
041B E6F0  ANI  OF0H
041D B1    ORA  C
041E 77    MOV  M,A
041F C9    RET

0420 7E    STH05: MOV  A,M
0421 E60F  ANI  HCHAR
0423 47    MOV  B,A
0424 79    MOV  A,C
0425 0F    RRC  
0426 0F    RRC  
0427 0F    RRC  
0428 0F    RRC  
0429 B0    ORA  B
042A 77    MOV  M,A
042B C9    RET

;******************************************************************************
; UTILITY
; INPUTS: C-ASCII CHARACTER
; OUTPUTS: CARRY=1 IF CHARACTER REPRESENTS VALID HEX DIGIT =0 OTHERWISE
; DESTROYS: A,F/F'S
; FUNCTION: VALDG returns success if its input argument is an ASCII character representing a valid hex digit, and failure otherwise.
;******************************************************************************
VALDL: MOV A,C

FUNCTION: VALDL returns success if its input argument is a valid delimiter character (SPACE, COMMA, CARRIAGE RETURN, EQUAL SIGN) FAILURE OTHERWISE.

INPUTS: C-CHARACTER
OUTPUTS: CARRY=1 IF INPUT ARGUMENT IS VALID DELIMITER CHARACTER =0 OTHERWISE
DESTROYS: A,F/F'S

UTILITY

;******************************************************************************
; INPUTS: C-CHARACTER
; OUTPUTS: CARRY=1 IF INPUT ARGUMENT IS VALID DELIMITER CHARACTER
; =0 OTHERWISE
; DESTROYS: A,F/F'S
; FUNCTION: VALDL returns success if its input argument is a valid delimiter character (SPACE, COMMA, CARRIAGE RETURN, EQUAL SIGN) failure otherwise.
;******************************************************************************

SONGN: DB CR,LF,'ADSL FM MUSIC SYNTHESIZER',CR,LF

CR,LF,CH,LF
CADDR: DW O
XCMD
SCMD
MCMD
GCMD
DCMD
RCMD
CCMD
ICMD
GCMD

CTAB: DB 'O'
DB 'I'
DB 'C'
DB 'W'

NCMDs EQU $-CTAB
RTAB: DB 'A'
ASAVE-((ASAVE/256)*256)
0
CSAVE-((CSAVE/256)*256)
D SAVE-((D SAVE/256)*256)
0
E SAVE-((E SAVE/256)*256)
F

EQU $-RTAB
DB 'B'
BSAVE-((BSAVE/256)*256)
0
CSAVE-((CSAVE/256)*256)
0
DSAVE-((DSAVE/256)*256)
0
ESAVE-((ESAVE/256)*256)
0
'F'
CALL BYTE
MOV L, A
CALL BYTE
ORA
JNZ RCM15
POP B
JMP RCM15
MOV C, E
CALL BYTE
MOV M, A
INX H
DCR E
JNZ RCM10
CALL BYTE
JNZ ERROR
JMP RCM05
CALL BYTE
JMP GETCM
MVI C, 2
CALL GETNM
POP D
POP H
MOV A, L
ADJ 10H
MOV C, A
MOV A, H
ACI O
MOV B, A
MOV A, E
SUB C
MOV C, A
MOV A, D
JC WCM10
MVI A, 10H
JMP WCM15
MOV A, C
ADJ 11H
**BYTE UTILITY**

**INPUTS:** D - CURRENT VALUE OF CHECKSUM

**OUTPUTS:** A - HEXADECIMAL CHARACTER

**D - UPDATED VALUE OF CHECKSUM**

**CALLS:** CI, CNVB

**DESTROYED:** A, D, F/F'S

**FUNCTION:** BYTE READS TWO CHARACTERS FROM THE CONSOLE AND CONVERTS THE CHARACTERS TO ONE HEX CHARACTER. THE A REGISTER CONTAINS THE FINAL CHARACTER AND THE D REGISTER CONTAINS THE UPDATED VALUE OF THE CHECKSUM.

****************************************************************************************************
055B C5 BYTE: PUSH B
055C CD8002 CALL Cl
055F E67F ANI PRTY0
0561 4F MOV C,A
0562 CD9302 CALL CO
0565 CD8A02 CALL CVNB
0568 07 RLC
0569 07 RLC
056A 07 RLC
056B 47 MOV B,A
056D CD8002 CALL CI
0570 E67F ANI PRTY0
0572 4F MOV C,A
0573 CD9302 CALL CO

0576 CD8A02 CALL CVNB
0579 B0 ORA B
057A 4F MOV C,A
057B 82 ADD D
057C 57 MOV D,A
057D 79 MOV A,C
057E C1 POP B
057F C1 RET

;****************************************************************************** PADR
UTILITY
*
INPUTS: HL-ADDRESS TO BE PRINTED
CALLS: PBYTE
*
DESTROYS: A
*
FUNCTION: PADR PRINTS THE ADDRESS IN THE HL PAIR BY CALLING PBYTE, THIS CAUSES THE CHECKSUM IN REGISTER D TO BE UPDATED.
*
;****************************************************************************** PADR: MOV A,H
0580 7C CALL PBYTE
0581 CD9905 MOV A,L
0584 7D CALL PBYTE
0585 CD9905 RET
0588 C9
PBYTE: PUSH PSW
RRC RRC RRC RRC CALL CALL POP PUSH CALL CALL POP PSW PSW PSW PSW PSW MOV D,A CALL BREAK JMP EXIT

PEOF

CALLS: CO,PBYTE,PAOD
DESTROYS: A,C,D,H,L,F/F'S
FUNCTION: PBYTE CONVERTS THE HEX VALUE IN THE A REGISTER INTO 2 ASCII CHARACTERS AND OUTPUTS THEM TO THE SERIAL PORT.
THE CHECKSUM VALUE IN THE D REGISTER IS UPDATED.

CALLS: CO,PBYTE,PAOD
DESTROYS: A,C,D,H,L,F/F'S
FUNCTION: PEOF PRINTS THE END OF FILE RECORD CONSISTING OF A RECORD MARK, A LOAD ADDRESS OF 0, THE RECORD TYPE, AND THE RECORD CHECKSUM.
05A6 0E3A PEOF:  MVI   C,:'':
05A8 CD9302 CALL  CO
05AC AF  XRA   A
05AD 57 MOV   D,A
05AE CD8905 CALL  PBYTE
05B0 210000 LXI   H,OH
05B3 CD8005 CALL  PADDR
05B6 9E01 MVI   A,1H
05B8 CD8905 CALL  PBYTE
05BB AF  XRA   A
05BC 92 SUB   D
05BD CD8905 CALL  PBYTE
05C0 C9 RET

05C1 0E0D PEOE:  MVI   C,CR
05C3 CD9302 CALL  CO
05C5 0E04 MVI   C,LF
05C7 CD9302 CALL  CO
05C9 C9 RET

05CC 4F4D4D41 CMSG: DB  'COMMANDS',CR,LF
05D0 4E44530B DB  'COMMANDS',CR,LF
05D4 0A DB  'C-COMMAND LIST',CR,LF
05D9 4D4D414E DB  'D-DISPLAY MEMORY',CR,LF
05E1 53540D0A DB  'G-EXECUTE PROGRAM',CR,LF
05E4 4D424449 DB  'I-INPUT FROM PORT',CR,LF
05ED 59204D45 DB  'O-EXECUTE PROGRAM',CR,LF
05F1 4D4F5259 DB  'T-TIMER',CR,LF
05F5 0D0A DB  'U-UNLOAD',CR,LF
05F7 472D4558 DB  'V-VIN',CR,LF
05FB 45435554 DB  'W-WIN',CR,LF
05FF 45205052 DB  'X-XEX',CR,LF
0603 4F475241 DB  'Y-YOUT',CR,LF
0607 4D0D0A DB  'Z-ZZONE',CR,LF
060A 492D494E DB  'U-UNLOAD',CR,LF
060E 50555420 DB  'T-TIMER',CR,LF
0612 46524F4D DB  'S-SUB',CR,LF
0616 20504F52 DB  'N-NAME',CR,LF
061A 540D0A DB  'M-MODE',CR,LF
0610 4D2D4D4D DB 'M-MOVE MEMORY',CR,LF
0621 5645204D
0625 454D4F52

0629 590DDOA DB 'O-OUTPUT TO PORT',CR,LF
0630 54555554
0634 20544F20
0636 504F5254
063C 000A
063E 522D5245 DB 'R-READ FILE',CR,LF
0642 41442046
0646 494C4500
064A 0A
064B 532D5355 DB 'S-SUBSTITUTE MEMORY',CR,LF
064F 42535449
0653 54555445
0657 204D454D
065B 4F52590D
065F 0A
0660 572D5752 DB 'W-WRITE FILE',CR,LF
0664 49544520
0668 46494C45
066C 000A
066E 582D4558 DB 'X-EXAMINE REGISTERS',CR,LF
0672 414D494E
0676 45205245
067A 47493354
067E 4552530D
0682 0A
0683 FF DB OFFH
0684 0000 DW 0

0686 D300 TBL1: OUT 0
0688 C9 RET
0689 DB00 IN 0
068B C9 RET

1.0 1.0
ORG EQU DB DB DB DB DB DB DB DB DB
ORG EQU DS DS DS DS DS DS DS DS DS
JMP JMP JMP JMP JMP JMP JMP JMP JMP
REGS 

; JUMP TABLE FOR USER RESTARTS AND INTERRUPTS

ORC C30000 R$1: JMP 0
38C6 C30000 R$2: JMP 0
38CG C30000 R$3: JMP 0
38CC C30000 R$4: JMP 0
38CF C30000 R$5: JMP 0
38D2 C30000 R$6: JMP 0
38DS C30000 R$7: JMP 0
38DB C30000 R$75: JMP 0
38DE C30000 TRAP: JMP 0

NO PROGRAM ERRORS
## SYMBOL TABLE

<table>
<thead>
<tr>
<th>SYMBOL</th>
<th>VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>0007</td>
</tr>
<tr>
<td>B</td>
<td>0000</td>
</tr>
<tr>
<td>BCHR</td>
<td>001B</td>
</tr>
<tr>
<td>BREAK</td>
<td>0267</td>
</tr>
<tr>
<td>BYTE</td>
<td>055B</td>
</tr>
<tr>
<td>C</td>
<td>0001</td>
</tr>
<tr>
<td>CM</td>
<td>0027</td>
</tr>
<tr>
<td>CNOUT</td>
<td>0030</td>
</tr>
<tr>
<td>CR</td>
<td>000D</td>
</tr>
<tr>
<td>CTCMD</td>
<td>00CF</td>
</tr>
<tr>
<td>CVNB</td>
<td>028A</td>
</tr>
<tr>
<td>DCM10</td>
<td>00CD</td>
</tr>
<tr>
<td>D</td>
<td>0003</td>
</tr>
<tr>
<td>ERROR</td>
<td>02C8</td>
</tr>
<tr>
<td>FRET</td>
<td>02D3</td>
</tr>
<tr>
<td>GCMD</td>
<td>00EC</td>
</tr>
<tr>
<td>GETNM</td>
<td>0311</td>
</tr>
<tr>
<td>GMN10</td>
<td>032D</td>
</tr>
<tr>
<td>GMN30</td>
<td>034F</td>
</tr>
<tr>
<td>GTC10</td>
<td>0066</td>
</tr>
<tr>
<td>HL05</td>
<td>0365</td>
</tr>
<tr>
<td>INUST</td>
<td>0368</td>
</tr>
<tr>
<td>KBRST</td>
<td>0008</td>
</tr>
<tr>
<td>LSAVE</td>
<td>3834</td>
</tr>
<tr>
<td>MODE</td>
<td>00CE</td>
</tr>
<tr>
<td>MHOUT</td>
<td>0387</td>
</tr>
<tr>
<td>PBYTE</td>
<td>0589</td>
</tr>
<tr>
<td>PRTO</td>
<td>007F</td>
</tr>
<tr>
<td>RBR</td>
<td>0002</td>
</tr>
<tr>
<td>RCMD</td>
<td>04C0</td>
</tr>
<tr>
<td>REGDS</td>
<td>03A4</td>
</tr>
<tr>
<td>RGADR</td>
<td>03D5</td>
</tr>
<tr>
<td>RS3</td>
<td>38C6</td>
</tr>
<tr>
<td>RS5</td>
<td>38D2</td>
</tr>
<tr>
<td>RSTOR</td>
<td>38C0</td>
</tr>
<tr>
<td>RTABS</td>
<td>0003</td>
</tr>
<tr>
<td>SCMD</td>
<td>018B</td>
</tr>
</tbody>
</table>

* X |
<table>
<thead>
<tr>
<th>SOMSG</th>
<th>008C</th>
<th>SP</th>
<th>0006</th>
<th>SRET</th>
<th>0400</th>
<th>SSAVE</th>
<th>3838</th>
</tr>
</thead>
<tbody>
<tr>
<td>STH05</td>
<td>0420</td>
<td>STHF0</td>
<td>0402</td>
<td>STHLF</td>
<td>040D</td>
<td>STLP</td>
<td>0065</td>
</tr>
<tr>
<td>STUP</td>
<td>0051</td>
<td>TBL1</td>
<td>0686</td>
<td>TEMP</td>
<td>383A</td>
<td>TERM</td>
<td>001B</td>
</tr>
<tr>
<td>TRAP</td>
<td>38DE</td>
<td>TRDY</td>
<td>0001</td>
<td>TXBE</td>
<td>0004</td>
<td>UPPER</td>
<td>00FF</td>
</tr>
<tr>
<td>USRBR</td>
<td>383D</td>
<td>*</td>
<td></td>
<td>VALDG</td>
<td>042C</td>
<td>VALDL</td>
<td>0447</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>WCM05</td>
<td>0510</td>
<td>WCM10</td>
<td>0525</td>
<td>WCM15</td>
<td>0528</td>
<td>WCM20</td>
<td>0540</td>
</tr>
<tr>
<td>WCM25</td>
<td>0555</td>
<td>WCMD</td>
<td>0509</td>
<td>XCM05</td>
<td>01D6</td>
<td>XCM10</td>
<td>01E5</td>
</tr>
<tr>
<td>XCM15</td>
<td>01F2</td>
<td>XCM20</td>
<td>0210</td>
<td>XCM25</td>
<td>0227</td>
<td>XCM27</td>
<td>0228</td>
</tr>
<tr>
<td>XCM30</td>
<td>0290</td>
<td>XCMD</td>
<td>01C4</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
REFERENCES


