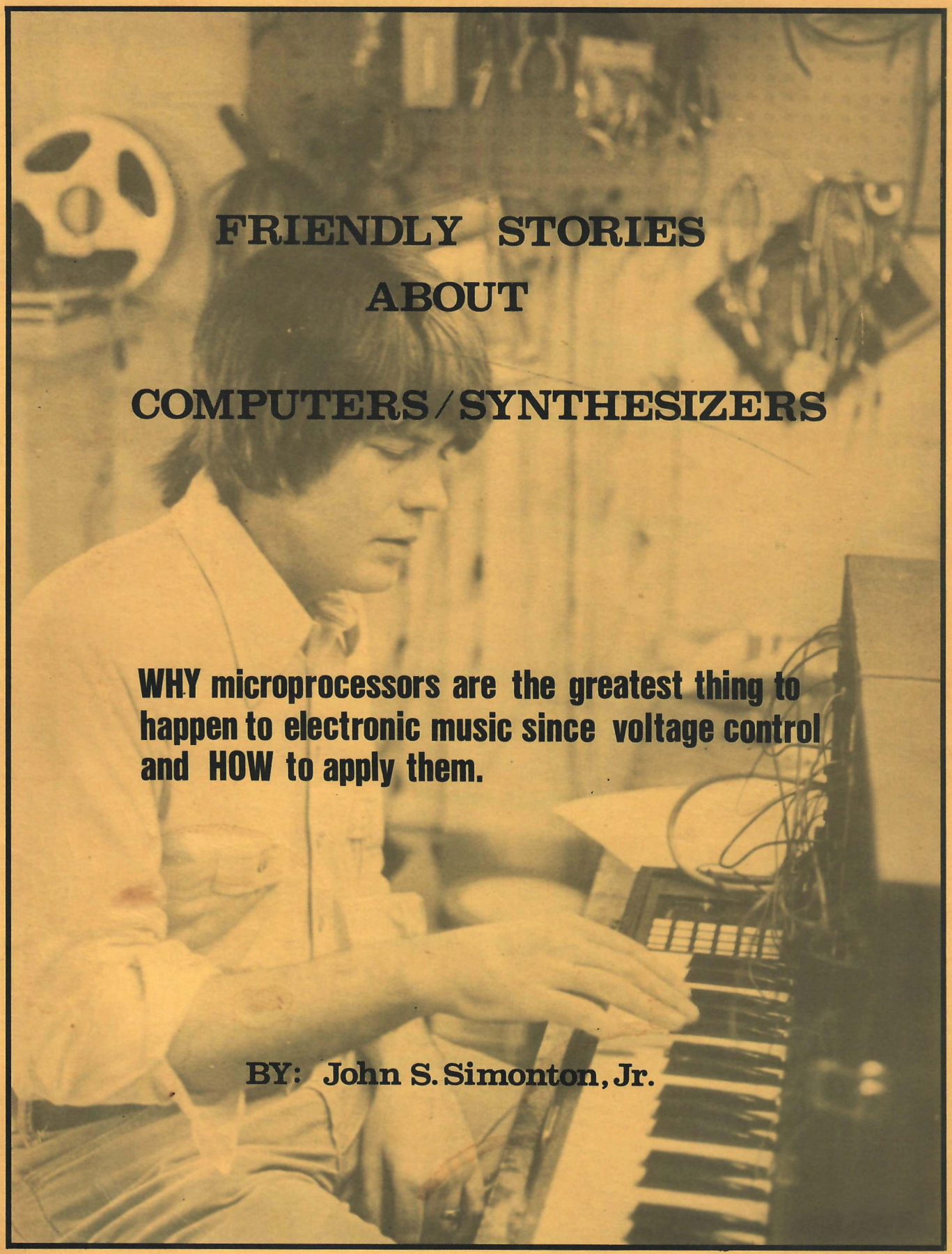


FRIENDLY STORIES ABOUT COMPUTERS/SYNTHESIZERS

WHY microprocessors are the greatest thing to happen to electronic music since voltage control and HOW to apply them.

BY: John S. Simonton, Jr.

FRIENDLY STORIES ABOUT COMPUTERS/SYNTHESIZERS (a design analysis)

BY: John S. Simonton, Jr.

Revised January, 1980

Published by:

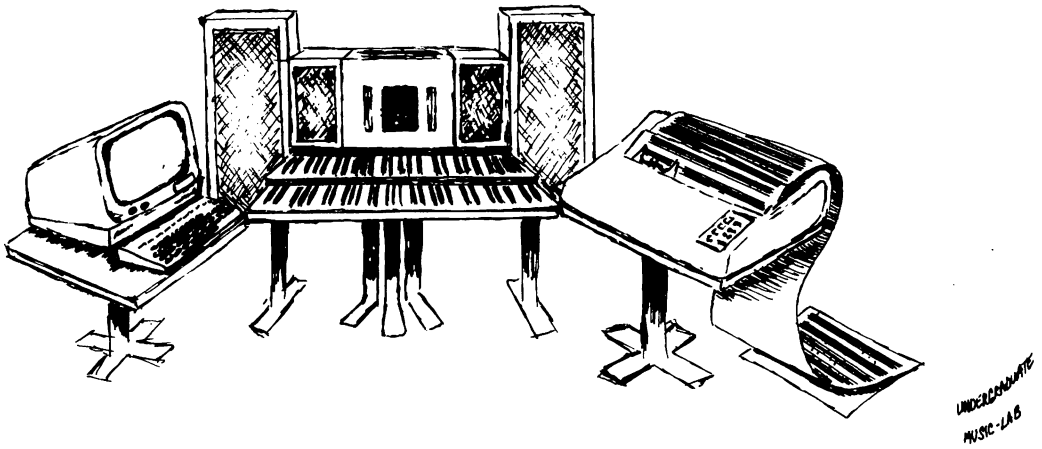
PAIA Electronics, Inc.
1020 W. Wilshire Blvd.
Oklahoma City, OK 73116

FRIENDLY STORIES ABOUT COMPUTERS/SYNTHESIZERS

CONTENTS

A Time Trip - READY:_

by: John S. Simonton, Jr.



I' m walking down a hall of oval cross section. Ahead and behind me the walls are colored by flowing patterns of blue, green, purple. As I pass, the sections of the walls closest to me burst into rapidly changing patterns of yellows and red. Softly, softly I hear a melodicly changing pattern of notes and chords. I notice these things no more than you noticed the color of your walls when you woke up this morning.

A sign inlaid in the wall announces:

UNDERGRADUATE MUSIC LAB

and I stop. In my mind I form a picture of a section of the wall sliding up and it does. I pass through the portal and picture the door closing.

Before me is a group of cabinets that obviously is a musical instrument. It's dominant feature is two stacked AGO keyboards. Above the keyboards, a panel with two central vertical doors about 1 by 6 inches flanked on either side by sections of grille cloth. To the right of the central console is a high speed printer and to the left a second keyboard, a typewritter keyboard, and above it a video terminal. In the upper left hand corner of the screen is written this word:

READY:_

I open one of the narrow doors in the contral console and insert a square cardboard jacket removed from a pocket in my assignment binder.

Sitting down, I type:

```
:LOAD "MUSE"
```

There is a barely audible click from the central console and after an instant' s delay a colon appears on the screen. I type again:

```
:PLAY "TUBA", K1, OUTL, OCT1
:RUN
```

and as I touch the lower AGO keyboard, fat juicy tuba notes come plopping out to the left speaker. Nice.

I type:

```
:PLAY "TUBA", K1, OUTL, OCT1
:PLAY "STRINGS", K2, OUTR, OCT5
:RUN
```

and now I get the tuba from the lower keyboard and left speaker while the right speaks phasy strings in response to my touch on the upper keyboard. Say, this is alright. I type:

```
:PLAY "DYNAMUTE", K1, OUTL, OCT3
:RUN
```

but this time a message displays on the screen. The message is:

ERR 10

Very cryptic. I remove a ring bound manual from a drawer below the keyboards. "PAIA 14700/S - Systems Manual" and I thumb through it until I find a section called "error codes". Here I find this entry:

ERR 10 Undefined Instrument Name.

Well, rats. I could have sworn that a simple thing like dynamute would have been in my instrument list. Too antique, I suppose; but fortunately it's a simple voice and I know it by heart, I type:

```
:DEFFN "DYNAMUTE":SIGNAL OSC(PULSE 1Ø),
 FILT(BP, Q5Ø, CC2), AMP (1ØØ): CNTRL ENVG
 (A1Ø, DØ, SØ, R5ØØØ) ( FILT, AMP), KBD (OSC)
 ;TRIG KBD(ENVG)
 ;PLAY "DYNAMUTE", K1, OUTL, OCT3
:RUN
```

Now as I play, the old familiar "wahp-wahp's" come from the speaker. A little trite perhaps, but still musicly useful in a piece that is to have an "old classic" sound to it. And just so I won't have to enter this voice again:

```
:INSTSAVE "DYNAMUTE"
```

The central console clicks. Now, to the real work. I type:

```
:SCORE "BASS1" C2/4, E2/4, G2/4, A2/4;
 R; TF2, R; TC2, R; TG2, R; TF2, R;
 C2/4, G2/4, F2/4, A2/8, C3/8, D#3/8,
 E3/8, C3/8, A#2/8, G2/2; BRIDGE, F2/4,
 F2/8, E2/8, F2/8, F#2/8, G2/4, D2/4,
 G2/8, F2/8, E2/8, D2/8;
```

Immediately the old familiar walking bass line "wahp's" its way into the room while I play string accompaniement on the lower keyboard. After diddling around for a while I come up with a melody line that I like OK and I type:

```
:SCORE "STR-LEAD", K1
:PLAY "STRINGS", "STR-LEAD", OUTR, OCT5
:PLAY "DYNAMUTE", "BASS1", OUTL, OCT3
:RUN
```

and play the lead that I liked. Now a moment to sit back and listen again. I type:

```
:PLAY "STRINGS", "STR-LEAD", OUTR, OCT5
:PLAY DYNAMUTE", "BASS1", OUTL, OCT3
:RUN
```

and everything that I played a moment ago is re-created. It sounds good but there's one note that's off. I type:

```
:LIST "STR-LEAD"
```

and the machine replies:

```
STR-LEAD: C4/8, G3/8, A#3/8,A#3/8, C4/16,
D#4/16, E4/16, G4/16, A#4/16, A4/16, G4/16,
E4/16, C4/8, G3/8, A#3/8, A3/8, C4/16, D#4/16,
E4/16, G4/16, A#4/16, A4/16, G4/16, E4/16
```

I can see what's wrong. That third octave A sharp in the first measure should have been a third octave A natural. I type:

```
:EDIT "STR-LEAD"
```

and the score is shown again but now there is a cursor at the end of the line. Using special keys on the keyboard I move the cursor back until it's under the error and then I press a key labeled "delete". The sharp is now a natural and with a PLAY instruction I have the line repeated. Now it sounds right.

Out of habit, more than anything, I type:

```
:COMPRESS "STR-LEAD"
```

and wait while the machine scans this score and reduces the memory space required by inserting "transpose and repeat" instructions wherever possible.

Using SCORE, EDIT and PLAY instructions I lay down another six tracks and then type:

```
:DEFFN "COMP1"
:PLAY "STRINGS", "STR-LEAD", OUTR, OCT5
:PLAY "DYNAMUTE", "BASS1", OUTL, OCT3
:-----etc.
```

and then:

```
:COMPSAVE "COMP1"
```

a click. And just to double check:

```
:CLEAR
:COMPLOAD "COMP1"
:PLAYCOMP "COMP1"
:RUN
```

It's not bad. There are only eight parts, of course, and it did take me a little longer than the graduate students; but they have modern Cyber-net activated instruments to use too. Having to bang away at the keys takes time. And in any case, it's all my work. I didn't use the HARMONY or CREATE instructions once. Poor old Dr. Biggle will like that. Now, before I shut it down:

```
:PRINT "COMP1"
```

and the high stacatto of the printer assures me that I will soon have a hard copy of the score on tablature.

I type:

```
:CLEAR
```

and the machine answers:

```
READY:_
```

WHAT THE COMPUTER DOES

The computer in our system does not itself generate any sound. It is simply acting as a performer/composer assisting control system for a more or less normal synthesizer. Providing what amounts to an extra set (or several sets) of hands.

From a system standpoint, it fits between the keyboard and synthesizer like this:

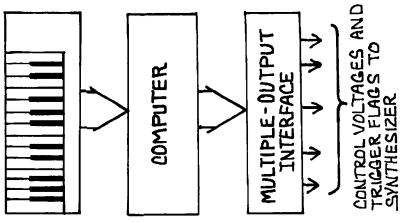


figure (a)

We said above, "more or less normal" synthesizer because there are three special elements involved in the synthesizer/computer interface:

1) a digitally encoded AGO keyboard (see "Computer music without the computer" and product summary)
2) a Digital to Analog Converter (see "Equally Tempered Digital to Analog Converter")
3) a multiple S/H circuit to allow several simultaneous outputs from the Digital to Analog converter.

The computer runs programs (either supplied by PAIA or user written) that receive data from the synthesizer keyboard and issue instructions to the D/A and multiple S/H which in turn control the synthesizer.

PROGRAMMING OVERVIEW

Just saying that the computer controls the synthesizer is hardly a satisfactory explanation of the system. Hardly satisfactory because it leaves out a

VERY IMPORTANT CONCEPT

which is that it is not really the computer that is controlling the synthesizer, it's the programs. In a very real sense, the computer is there only because it's a way to run the programs.

One of the programs (for example) "reads" the synthesizer keyboard and builds a table of what it finds there.

If the phrase "builds a table" is unfamiliar to you, it simply means that when the program finds that a given key is down on the keyboard it records in a special place (location or address) in memory which key it is. The next key that it finds down, it records in the next memory location; and so on. When the program has finished looking at the entire keyboard the result is a list or "table" of the keys that were down during that scan. If you were holding down a C chord for example, the table might look like this:

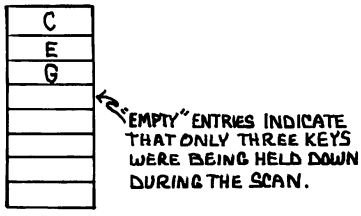


figure (b)

That's not really all there is to this program - there are some subtleties that would probably be confusing at this point. We'll get to them later. For right now, we'll just think of this program as a list-builder.

Also, so that I won't have to keep typing "the program that builds the list of keys that are down on the keyboard", we'll agree among ourselves that we'll call this program by the name "LOOK". From now on, when I say something like "we LOOK at the keyboard" you'll know that I mean we "execute" (run) this program.

And, while we're hanging labels on things, we may also just as well name the list that LOOK generates "key-table", or, since I'm a lazy typist, just KTABLE.

Got that? LOOK builds KTABLE.

OK, next.

There is another program that we'll call NOTEOUT, because it takes care of outputting the notes.

Like LOOK, this one can be stated in simple terms: it reads the first entry from a table and causes the D/A to convert that key data to a control voltage which it then strobes into the first S/H. It then gets the second entry from the table, converts it to a control voltage and assigns it to the second S/H. Gets the third entry, etc.

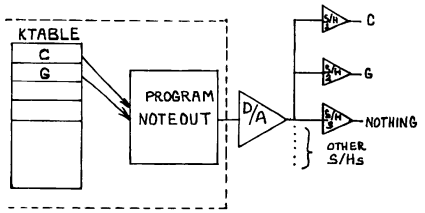
Also, like LOOK, there are subtleties that we'll look at later but the important point is that this routine works quickly. A block of 32 Sample and Holds can easily be refreshed and up-dated in about 16 ms. - more than fast enough.

The table that is read by NOTEOUT we will call the "note-table" or, simply NTABLE.

LOOK builds KTABLE and NOTEOUT reads NTABLE. Maybe you're wondering why two tables - why not just one.

Well, we could do it that way - if we did, a simplified diagram of the system should look like figure c.

You will recognize that we're still holding down that C chord. Now suppose we let the E go. On the next scan of the keyboard, LOOK up-dates KTABLE to reflect the fact that the E is no longer held down. KTABLE now looks like this:



THIS MIGHT NOT BE TOO BAD — MANY ORGANS DO NO MORE.

figure (d)

And when NOTEOUT reads this table and up-dates the S/H circuits, guess what? The G has moved to the loca-

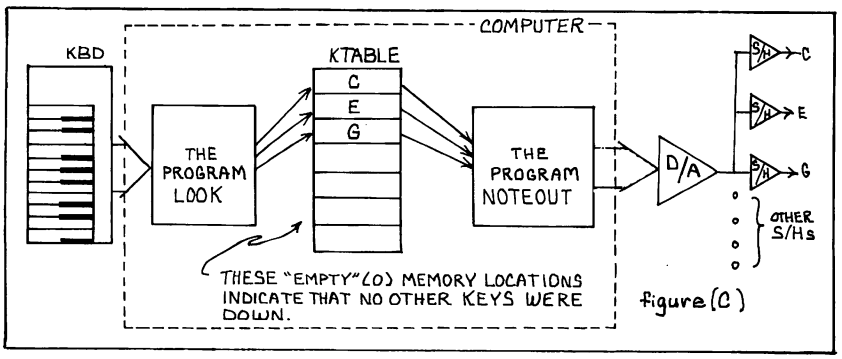


figure (c)

tion previously occupied by the E and from the S/H that previously was producing the control voltage for the G we now have ..... nothing.

As if it weren't bad enough that the VCO which was previously producing an E is now playing a G (and we can hear when it makes this change), we can't do any decay processing on the E - the way a natural instrument would - because it's not there anymore.

Maybe this isn't too bad. A lot of organs produce results very similar to this - and all multiple output analog keyboards do this exact same "guess where the note's going to come out" trick. Still, it seems that there would be a more pleasing way to do it.

There is.

Because we're using two tables, we can generate a large (very large) family of programs that make decisions on how to transfer the information from KTABLE to NTABLE. This produces a machine which diagramaticly might look like this:

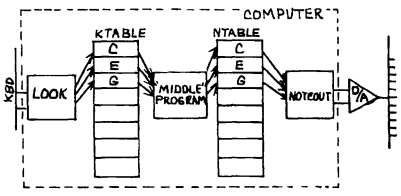


figure (e)

How this new middle program makes transfers from KTABLE to NTABLE determines completely the "personality" of the instrument.

For instance, a better way to handle the multiple -output problem would be to have the "middle" program not delete an entry from NTABLE simply because it no longer appeared in KTABLE, but rather to indicate that while the note should still be played, the key corresponding to it was no longer being held down and decay processing should begin. This is where the concept of "flags" associated with each note comes in and while it is slightly out of sequence, we should examine this important feature now.

The data that goes out to the synthesizer interface is a collection of 8 binary digits (bits - "1" or "0"). Like this:

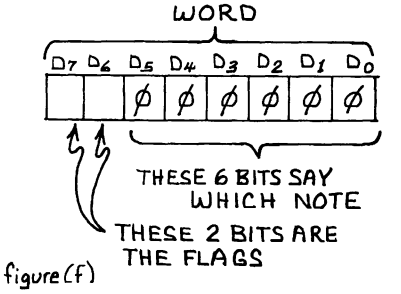


figure (f)

If we want to indicate to the synthesizer that the note that the data represents is one which currently corresponds to a key that is being held down on the keyboard, then we set bit #7 (D_6) to a "1". If the data does not correspond to a key that is currently down then this bit is a zero. As you can see, if you're already familiar with synthesizers, this flag bit corresponds to the "gate" signal that you get out of most synthesizer keyboards.

As you will see when you review the included 8780 information, both of these higher order bits are buffered and brought out to the front panel of the Equally Tempered Digital to Analog Converter.

This leaves us with a "left-over" flag that can be used in a variety of ways. It can, for instance, be used simply as an independent gate signal allowing the processor to select between one of two patching arrangements that we've set up. Or, and I believe that this is the preferable use, it can be used as a GLIDE SELECT bit that turns glissando on and off - under computer control.

But, to get back to the real subject at hand, the polyphonic output procedure described above is not the only (or, in my opinion, the most) interesting thing that the "middle" program can do.

It can examine the entries in KTABLE and if they are lower than a given note on the keyboard assign them to one group of outputs and if they are higher assign them to a second group of outputs. Which has the effect of "splitting" the keyboard into two different voices - one for low keys and a second for high keys.

The "middle" program can take notes from the keyboard and not only play them immediately, but also store them in another permanent table in the machine's memory for playback again later.

The "middle" program can take notes from the permanent table mentioned above, assign them to outputs and simultaneously assign current keyboard activity to other outputs - so that you can play along with something that was previously "recorded".

These same programs can allow independent recording and simultaneous playback of multiple "tracks". Like a multi-track recording studio only without the hassle of tape splicing, editing and (worst of all) over-dubbing noise.

The "middle" program can do tricks like making a chord played on the keyboard seem to be rising in pitch, constantly, without ever actually going beyond a pre-defined limit. It's not magic, it involves forming a "stack" of the notes and allowing the program to increase the pitch of the notes in the stack until they reach a pre-determined limit at which time the note is "faded out" and placed in the bottom of the stack.

The "middle" program can do lots of different things. So many, that it's going to be a while (possibly a long, long while) before we know what they all are.

If you're looking for something that will reach a "finished" state beyond which there is nothing further to do, this isn't the product for you.

SO MANY "DIFFERENT" PROGRAMS

One thing that you may notice in the discussion above is that all of these very different "resource allocation" schemes have in common the fact that they all use LOOK and NOTEOUT. We could make these two routines a part of each of the larger programs if we wished - there wouldn't be any problems with that - except that they are long-ish and would take a while to "load" into the machine's memory. Particularly if you're not using the computer's optional cassette interface. I think there's a much better way.

We can write the LOOK and NOTEOUT programs so that they're what's known as "subroutines".

Now ordinarily, computer programs proceed sequentially through memory an instruction at a time. Like this:

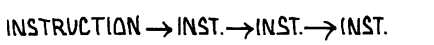


figure (g)

But a subroutine allows a block of programming to be stored out of sequence in the machine so that when you "call" or "jump to" a subroutine it's like this:

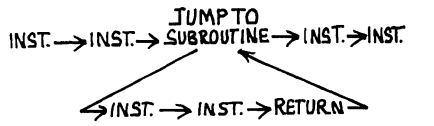


figure (h)

The "return" causes the computer to go back to the place that it was before the subroutine was called and continue executing the main program.

Maybe the "subroutine" concept confuses you (though after such a terrific explanation it's hard to imagine how). If it does, here's another way that you can think of them:

SOFTWARE MODULES

You're certainly used to synthesizer "hardware" modules by now - all those little processing elements (VCO's, VCF's etc.) that we tie together with patch cords to produce different sounds or effects.

Here we have their equivalent in computer instructions - little modules of programming that are patched together (not with wire, of cource, with more programming) which, depending on how they're tied together, produce different effects.

LOOK and NOTEOUT are not the

only software modules that are useful, others include SAVE (the "recording" module, SREPRO (the "playback" module), DELAY (a time delay routine), POLY (a useful polytonic resource allocation algorythm), and others.

These various modules are available in a number of different forms. They're available just as program listings (which can be manually entered into the computer - very tedious but about as cheap as you can get) or they're also available on cassette tape that can be loaded into the computer using the optional cassette interface.

First choice for a place to save these universally useful programs, though, is Read Only Memory.

This is the most expensive alternative (ROMs have to go for about $20/each - one would be filled by the programs mentioned above) but it has the advantage of NOT HAVING TO LOAD THE PROGRAMS AT ALL. Every time you turn on the machine, they're there, waiting to be used.

SOUNDS INTERESTING WHAT DO I NEED TO GET STARTED?

If you already have some PAIA synthesis equipment, you're well on the way, but you need to convert to the new digital format. We've tried to make that as easy and inexpensive as possible by providing a retro-fit kit to digitally encode your present PAIA keyboard, the EK-3 Keyboard Encoder Kit mentioned in the POLYPHONY "Lab Notes" reprint included in this package.

This encoder is primarily designed to fit 4700 series keyboards, but will of course fit 2720 series equipment as well. It is one of our experimenter's kit series and does not include step-by-step instructions. In fact, the EK-3 re-print that is part of this package is the instruction set.

If you want to start over with a new keyboard, we have the 8782 Encoded Keyboard - one of our full kits with complete instructions.

If you already have an organ and would like to use that keyboard for either synthesizer or synthesizer/computer interface, we have the EK-4 Organ Keyboard Encoder as described in the accompanying package.

The advantage to this is that the keyboard already in the organ may be used for both synthesizer/computer and organ - all at the same time. Even if there are no "spare" contacts on the keyboard.

BUT I DON'T HAVE A SYNTHESIZER!

Looking back over the text to this point I notice an important point that has not been prominently mentioned. This system - because of the properties of the D/A - will work only with low-cost LINEAR synthesizer modules. Synthesis modules whose characteristics are exponential cannot be used (though it is an easy matter to substitute another D/A for ours).

It is difficult to tell someone what the configuration of their synthesizer should be. Particularly with modular equipment like our current line. The modules that make up the system are so much a function of the use to which the system is to be put.

Never the less, we have two systems configured as starting points. "Starting points" because it has been our experience that most people add and make changes to their system as time goes on. Customizing it to their application.

These two packages are the 4700/C (primarily a monotinic system) and the 4700/J (suitable for polyphonic work, limited multi-track recording, etc.). These are both systems that we originally put together to take to shows. Each for its intended purpose, they have proven to be reliable and versatile; each capable (by design) of turning someone from an "I don't like synthesizers" person into a "I never realized they could do that" person. Maximum usefulness and versatility within minimum "waste" capacity.

The module complement of each of these systems is itemized in the product summary, but this would seem an appropriate place to discuss the "philosophical" (if you will, just this once, excuse so pretentious a term) implication of the systems.

The 4700/C is a minimal, useable system. It has roughly the capabilities of the "mini" this and that that you see advertised. It's made for people who find synthesis interesting but aren't really sure that they're going to get into it in a big way. It is (briefly) an ideal place to start. And since all of our gear is modular and available separately, it is a system which will easily grow as your interest grows.

The 4700/J is by the standards of the industry a "good-sized" system. It's difficult to make comparisons, since some of the modules (particularly those that are the computer interface) aren't available from other manufacturers; but, if these modules were available and you purchased them assembled through the normal distribution chain the '/J would be on the order of $2,500 to $3,500 worth of equipment. And, again, it's not a dead-end system, but one that can grow.

One final comment in this section is in order, and it may seem strange for someone who is, after all, trying to sell you equipment:

DON'T OVER-BUY

There are two reasons for making a statement like this - both imminently practical; 1) our experience has been that you will probably like the equipment a lot and will be a customer for many years, but if you don't (and aren't) you don't have a bunch of money sunk in something you're not going to use. We won't have someone wandering around bad-mouthing the gear.

2) Without committing to anything in print, development goes on all the time - to the practical synthesist, the versatility of modular equipment makes it desirable to have some of it around (ask anyone seriously involved in electronic music synthesis). But, well, look at any issue of POLYPHONY - development it goes on and you never can tell what's just around the corner.

WHICH COMPUTER?

This one is almost as bad as which synthesizer. For the same reasons - the decisions are very personal and user related. Also like the "which synthesizer?" though, we have suggestions.

Our first, and strongest, suggestion is our own 8700 Computer/Controller. High on the list of compelling reasons to select this machine should be the fact that it will have our fullest software support (all of the programs mentioned earlier are available now), it is physically designed to fit into a space that has been kept free in our 4700 and 8700 series keyboards and is a machine designed to the PAIA ideal of "maximum impact for minimum bucks".

The 8700 is based on a 6503 processor (a fully software compatible version of the increasingly popular 6502) and has features as described in the product summary. This processor was chosen over others which were - at the time that the decision was made - more popular for a variety of reasons, but by far the biggest was that it is an easy machine to use. Even if you're programming in machine language (and don't kid yourself, the day will probably come that you will want to do something completely different - something not available either from us or from the independent user's group program exchange - and the only way to do it will be to write the code yourself, it's easier than it looks).

But let's suppose that you already have a computer. If that computer happens to be something like a KIM-1, you're in great shape. We will shortly have a complete KIM-1 package showing how to interface and almost as complete a selection of programs as for our own machine (we like the KIM series stuff - and since it, too, uses a 6502.....)

If you have a SWTP 6800 system, the 8780 and 8782 instructions already outline using one of their MP-L's for

interfacing (sorry, no software support from us right now, but surely the user's group will come up with some - Southwest has a really nice, popular system).

Coincidentally, there are other machines that use the 6502 processor for which all of our software is written; if you haven't heard of them yet, you will.

They are:

Commodore's PET (personal electronic transactor) which looks at this point like it will sell in the $600.00 range. Certainly you're all familiar with Commodore - they're an old-line (if there is such a thing) calculator company.

and

Apple Computer Company's APPLE II

We like the APPLE II machine a lot and probably a single glance at the enclosed literature will tell you why. It not only looks nice and can grow up to be a VERY LARGE system, but it has all the bells and whistles including FULL-COLOR VIDEO GRAPHICS capabilities (vectored, no less). I own one (one of the very first, I'm led to believe) and I can tell you - it's a very impressive system.

Equally Tempered Digital to Analog Converter

By: John S. Simonton, Jr.

Many experts will tell you that in order to interface a computer to an electronic music synthesizer, you must use exponential response voltage controlled elements (oscillators, filters, amplifiers, etc.).

Here's why:

Computer control of synthesizers requires a Digital to Analog converter to change the numbers that the computer puts out into an analog control voltage that the modules can use.

By far the most common type of D/A (so common that many seem to think it's the only kind) is known as an "R/2R ladder". I don't want to get into the design details of this circuit. If you are interested, there is plenty of information available from text-books, manufacturers literature, etc. But we do need to examine a functional aspect of this circuit.

Any analog to digital converter works by accepting at its input a digital quantity (we will call this data) and generating at its output an analog voltage that is a unique representation of that data. Most of the D/A's that I'm familiar with accept the data as binary digits - a bunch of 1's and 0's that appear simultaneously on a group of wires going into the converter.

In a R/2R ladder converter, a unique weighting is assigned to each bit in the data coming in. When the time comes for a conversion to be made, the circuitry adds together the weightings corresponding to the bits in the data that are in an "on" state (for our purposes, a 1; through not always) and ignores the weighting represented by the bits that are "off" - equivalent to adding in a zero.

If we assume that we are going to be using exponential response oscillators, the R/2R ladder converter works quite well. We can assign weightings to the bits that are integral multiples of 1/12 volt; the same incremental voltage change that keyboards designed to operate exponential oscillators produce, and when we do we come up with a series of weightings which - progressing from the Least Significant Bit (LSB) to the Most Significant Bit (MSB) - Looks like this:

LSB **MSB**

$$1/12,\ 2/12,\ 4/12,\ 8/12,\ \ldots\ldots\ n^2/12$$

Figure 1

Where n is, of course, the number of bits that the converter can accept as data.

Let's watch four bits "count" into this type of converter and observe the resulting output voltages.

DATA	MEANS	OUTPUT
0 0 0 0	0+0+0+0	0
0 0 0 1	$0+0+0+1/12$	$1/12$
0 0 1 0	$0+0+2/12+0$	$2/12$
0 0 1 1	$0+0+2/12+1/12$	$3/12$

------		
1 1 1 1	$8/12+4/12+2/12+1/12=15/12$	

Table 1

If I had made the "word" (collection of 1's and 0's going into the converter) 6 or 8 bits long instead of just 4, the resulting series of output voltages would still increase 1/12 volt for every unit increment of the data and the only effect would be to increase the range of the output voltage.

Unfortunately, while the distinguishing feature of an exponential oscillator is that equal incremental voltage changes will cause it to generate a series of equally tempered pitches, this is not the case for linear response oscillators. A linear oscillator requires constantly increasing voltage increments to produce equally tempered semi-tones.

While this increasing voltage requirement doesn't make the application of R/2R converters to linear oscillators impossible, it certainly makes it cumbersome.

Cumbersome because we have to make the incremental change from the converter small enough to guarantee that there will be some pattern of 1's and 0's that defines a control voltage reasonably close to what we're really after.

Very small voltage increments - there are three things "wrong" with this:

1) We're going to need a "bigger" converter - one with greater resolution and consequently greater word size. Whereas 6 bits of data will provide a little more than 5 octaves of control voltage to an exponentail oscillator; the same 5 octaves from a linear oscillator will require 12 data bits. Now, if that doesn't offend you by its notable lack of elegance, it's cost certainly should. A 12 bit D/A is going to set you back about $100.00; then you've got to put it on a pc board, add controls - front panel, etc.

2) As if to add insult to injury, there will be lots of combinations of bits that represent the intervals between adjacent semi-tones, but notice that they are not equally tempered intervals and therefore next to useless even for micro-tonal tunings. We're paying out our hard earned bucks for words that we're never going to use, but must have to fill up the "cracks".

3) We've turned the determination of what data to output from a relatively simple matter of counting the keys and using that as the data into a process that at best is going to require a look-up table (where the machine says "Aha - key number 12, that's note 0001110010100001") or some such similar computer calisthenics. Not particularly complicated, perhaps, but why bother with it if we don't have to.

And that, friends, is the point of all this. We don't have to. For the simple reason that an R/2R ladder converter is not the only kind that we have to work with. There are other kinds. One of the other kinds is called a Multiplying D/A (or just MD/A, I guess).

While the most important operational feature of the R/2R ladder converters was that it added things to arrive at the output, the dominant feature of an MD/A is that (you're ahead of me, right?)

IT MULTIPLIES.

Far out.

If you're up on your basic music theory, a responsive chord (if you'll pardon the expression) should be struck here. The determination of the frequency of the pitches in equally tempered tunings is itself a multiplication process. The frequency of each semi-tone in the series is greater than the frequency of the preceding semi-tone by a factor of $2^{1/12}$ - the infamous "twelfth root of two" ($2^{1/12}=\sqrt[12]{2}=1.059$). Intuitively, it would seem that this type of D/A would be more appropriate for our purposes.

In fact, this is true. We assign weightings to the bits (starting with the LSB) according to this series:

LSB MSB

$$2^{1/12},\ 2^{2/12},\ 2^{4/12},\ 2^{8/12}.....2^{2^n/12}$$

Figure 2

Where again, n is the number of bits of data that the converter will accept.

Now, we count into this converter the same way that we did in the R/2R ladder type. Remember that bits that are "off" here are not included in the total (only now this is equivalent to multiplying by 1) and that the product that results from the condition of the data will be multiplied by some internal reference voltage.

DATA		OUTPUT
0000	$1\cdot1\cdot1\cdot1\cdot V_{ref}$	V_{ref}
0001	$1\cdot1\cdot1\cdot2^{1/12}\cdot V_{ref}$	$2^{1/12}V_{ref}$
0010	$1\cdot1\cdot2^{2/12}\cdot1\cdot V_{ref}$	$2^{2/12}V_{ref}$
0011	$1\cdot1\cdot2^{2/12}\cdot2^{1/12}\cdot V_{ref}$	$2^{3/12}V_{ref}$

1111	$2^{8/12}.2^{4/12}.2^{2/12}.2^{1/12}.V_{ref}$	$2^{15/12}V_{ref}$

Table 2 *

* Multiplying a base number raised to various powers (exponents) is accomplished by adding the exponents. That's how a slide rule works - remember slide rules?

You may recognize this as an equally tempered series (if not you'll just have to take my word; it is). All we have to do now is design a circuit that does this.

Let's do that.

Here's a simple unity gain buffer amplifier:

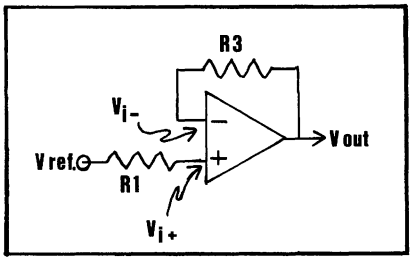


Figure 3

You may not be used to seeing it in this form because ordinarily the resistances that are shown would be replaced with direct connections. But having the resistances there doesn't matter simply because for any practical case, they are going to be much smaller than the equivalent resistance from either of the operational amplifier's inputs to ground. I should mention here that for any linear operational amplifier circuit the voltages at the inverting and non-inverting unputs are equal ($V_{i+}=V_{i-}$; this is the key to op-amp design, but that's another story). Of the circuit in figure 3 we can say:

(a) $$V_{out} = V_{ref}$$

An excellent beginning. Here's another circuit:

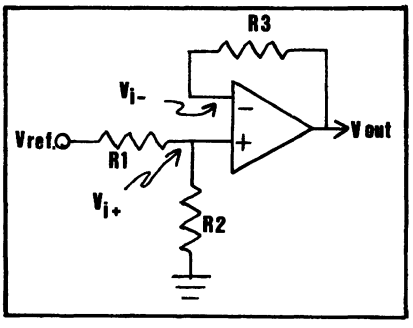


Figure 4

Adding R2 to the circuit has produced a voltage divider at the + input of the op-amp and because of this we can say:

(b) $$V_{out} = \left(\frac{R2}{R2+R1}\right) V_{ref}$$

Fantastic. Now we change the circuit again so that it looks like this:

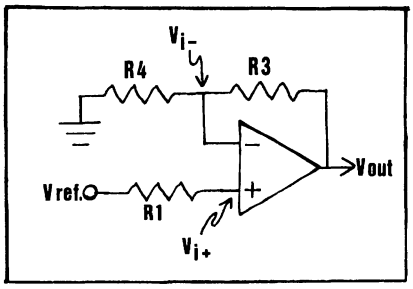


Figure 5

and instead of a voltage divider at the + input we now have one at the - input. This means that:

$$(c) \quad V_{out} = \left(\frac{R3+R4}{R4}\right) V_{ref.}$$

All together now:

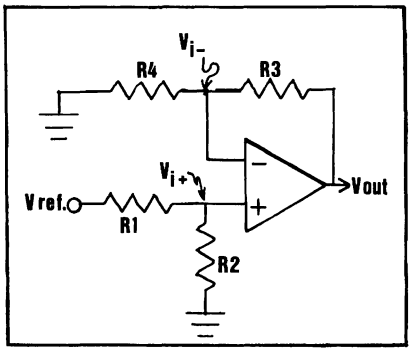


Figure 6

And for this configuration:

$$(d) \quad V_{out} = \left(\frac{R2}{R1+R2}\right)\left(\frac{R3+R4}{R4}\right) V_{ref.}$$

Do the four equations from (a) - (d) look familiar? No? Look back at Table 2. Now do they look familiar? Still no? Then let's say:

$$(e) \quad \frac{R2}{R1+R2} = 2^{1/12}; \quad \frac{R3+R4}{R4} = 2^{2/12}$$

and then by making these substitutions and putting the equations together:

(a) $V_{out} = V_{ref.}$ Ⓐ

(b) $V_{out} = 2^{1/12} \cdot V_{ref.}$

(c) $V_{out} = 2^{2/12} \cdot V_{ref.}$

(d) $V_{out} = 2^{1/12} \cdot 2^{2/12} \cdot V_{ref.} = 2^{3/12} V_{ref.}$

Now you must certainly recognize them - it's the same series as the first four entries in Table 2. Putting the resistors R2 and R4 into the circuit and removing them is simply a matter of putting switches (either mechanical or electronic) in series with them and when we do the whole circuit looks like this:

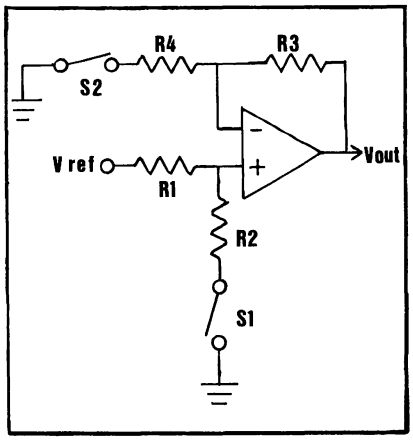


Figure 7

The switches S1 and S2 here are, respectively, the Least Significant and Most Significant data inputs to the converter; and I will avoid the obvious comment about this being a 2 bit D/A.

Oh, but there's one thing that I forgot to tell you:

$$(f) \quad 2^{1/12} \neq \frac{R2}{R1+R2}$$

Why? Because $2^{1/12}$ is a number greater than 1 and the only way that the ratio of a number to itself plus something else can be greater than 1 is if the something else is negative - which in our case, it's not (yes, there is such a thing as negative resistance, but the concept is not applicable here).

Happily, we have an alternative to negative resistance and that is to make:

$$(g) \quad \frac{R2}{R1+R2} = 2^{-1/12}$$

Making the exponent negative is equivalent to taking the reciprocal of the number.

At this point I'm afraid that in the interest of brevity I must make a gigantic leap and say that -- because we're using the reciprocal of the weighting, we must also complement the bit representing that weighting. In the instruction manual for this module, we will cover why. But there's not enough space to do it here. And, in any case, any of you who really want to can figure it out for yourselves. It's easy, honest.

Expanding this D/A out to handle greater word lengths is simply a matter of cascading several of these sections. When we do this and replace the mechanical switches that we had earlier with 4066 type Quad Bi-lateral CMOS switches we come up with a thing that looks like the circuit shown in figure 8.

Notice that the complemented bits that we require are indicated by the overbar on like $\overline{D_0}$ for instance. This is read "not D_0" and by custom indicates that the low (0) state is considered to be "on".

You are probably also wondering about those R_a's, R_b's, etc. The values of these resistors are determined by solving equations (a) through (d) and they produce some strange values that need to be exact. 5%'ers won't get it here. In order to meet the necessary precision and stability requirements, we've had "one of the nation's leading resistor manufacturers" (at least that's what they say) make up some custom Cermet resistor networks. They look about like any 16 pin DIP IC (except that they're a beautiful robin's egg blue), but inside are resistors instead of other stuff. Once manufactured, they're trimmed by LASER to be exactly the right ratios (Laser, yet - how about that!).

I really don't expect that to impress you too much, but this should:

THERE ARE NO ADJUSTMENTS TO THIS MODULE

You just put it together and it "plays" (which is the computer people's phrase for works).

Do you realize that this gets rid of all those trimmers from our old '-8 keyboard - it even gets rid of the zero pot. I really like it.

But we're really not through yet, we need to completely dress the design by adding input latches (so that the input information can be stored), and some kind of indicators so that we will know what's going on (LED's - they wink, they blink, they twinkle like stars in the night; anybody can look at this thing and know that it's got something to do with computers). This part of the circuit is shown in figure 9.

The 4042's are the latches and one of their features is that they have both Q and $\overline{Q}$ (the complement of Q) outputs- since we needed some complemented data bits, this is nice. Q9 - Q14 are level converters. We need to have the "on" resistance of the 4066 switches in the converter circuitry working at as high a supply voltage as possible in order to achieve predictable low "on" resistances and this means that they operate from the +9v. synthesizer supply rather than the +5v. logic supply.

That's the design. Let's take a few minutes to review what we've got here.

We've got a new synthesizer module that does at least one thing that many people thought couldn't be done; a 6 bit Digital to Analog converter that will provide slightly more than 5 octaves of equally tempered control voltage to linear response voltage controlled synthesizer elements.

The front panel PITCH control allows the module's output to be chromatically transposed over another octave, so the total range of output voltage available is a little more than 6 octaves (compared to typicly 4 octaves for a #4782 keyboard).

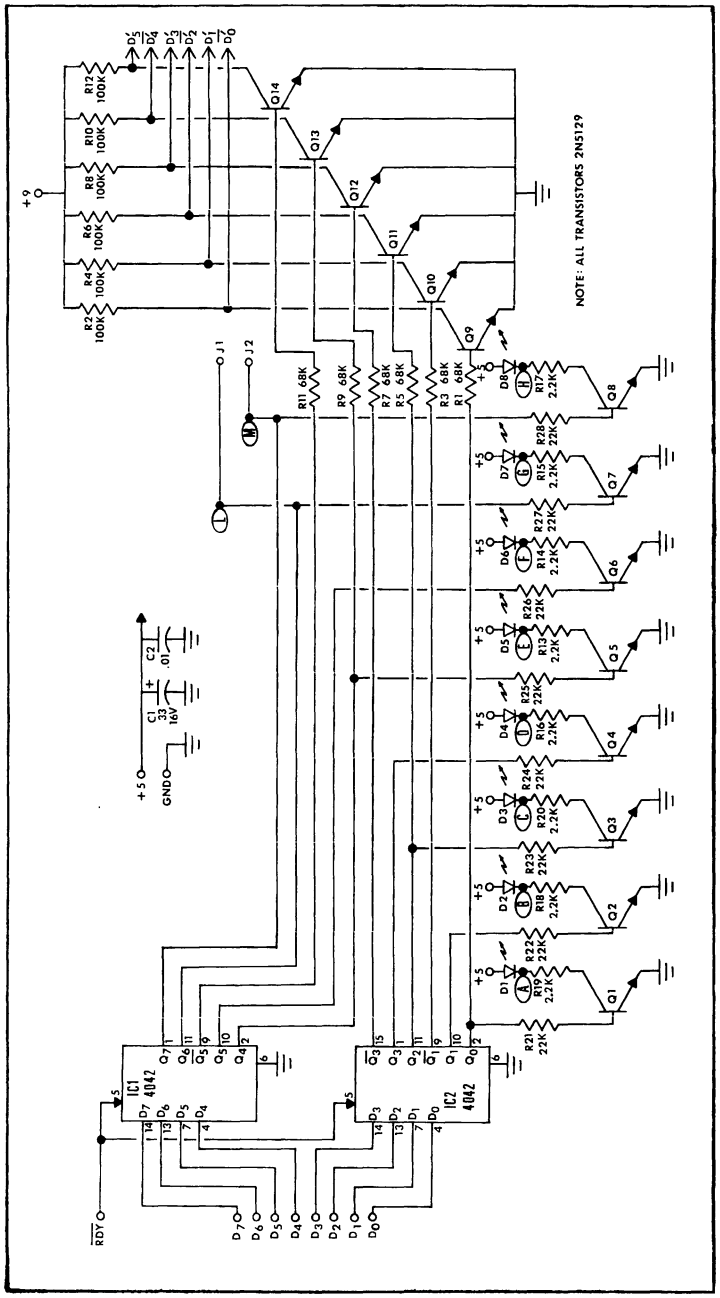
We have two trigger flags available, either of which can be set or re-set independently (very handy). As we will see in a future issue, these flags can also be used to select micro-tonal intervals.

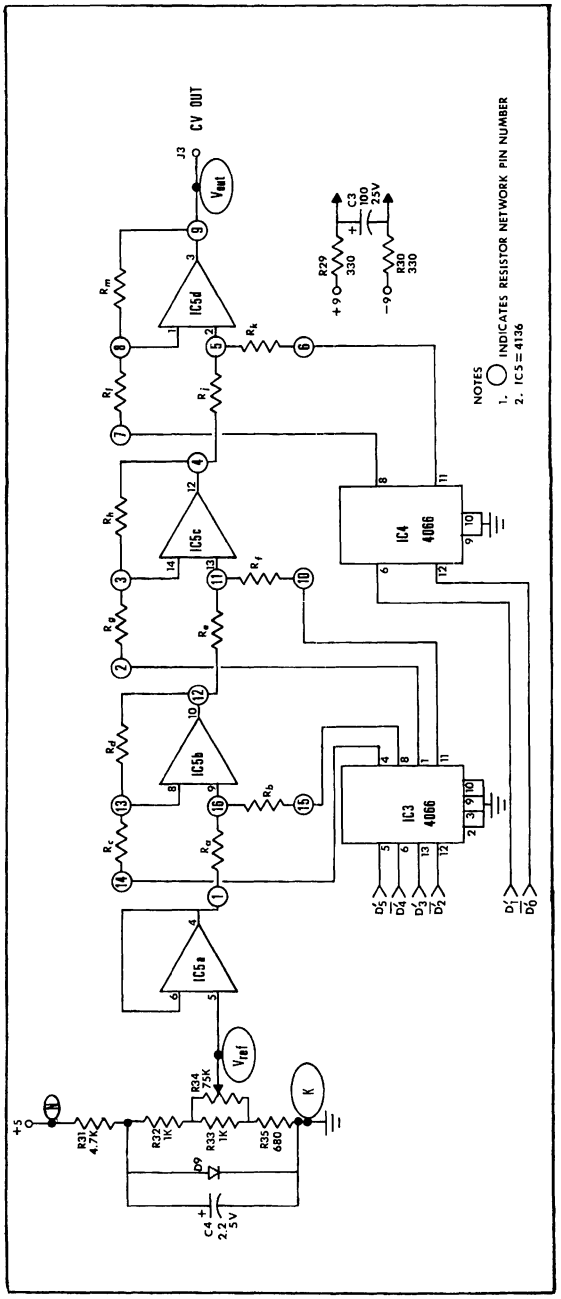
The status of the 8 bits of data coming into the module is displayed on the front panel LED's, six of which indicate the note that the module is producing and two of which indicate the status of the trigger flags.

To make the module easy to interface to anyone's computer (or simply keyboard encoders - see LAB NOTES) we have an input terminal marked $\overline{\text{RDY}}$ (not ready). When this terminal is grounded, the latches that are provided on the data lines are in a "pass" condition and any changes of the data on the data input lines will be reflected as changes in the module's output voltage. When the $\overline{\text{RDY}}$ line is taken to a high logic state, the last data that appeared on the input lines is stored in the latches and further changes on the data bus will not produce any change in the output voltage (this is about like the action of the SAMPLE inputs of clockable sample and holds).

The road to applying the processing and control power of the computer to electronic music synthesis is not a short one - but it is certainly a trip worth taking. The Equally Tempered D/A is only a first step.

As first steps go, though, this is a good one - like walking in seven league boots.





LAB NOTES

COMPUTER MUSIC, WITHOUT THE COMPUTER. or: What to do 'til your processor arrives.

By: John S. Simonton, Jr.

I realize that a lot of you will respond to the introduction of the 8780 Equally Tempered D/A with a frustrated, "But, I don't HAVE a computer."

Here's a little surprise. You don't really need a computer to do some very interesting and useful things with the 8780. You are going to need some additional hardware, as we'll see in a moment, but it's not only inexpensive, it's also equipment that you'll need for processor interfacing later on anyway. You're not building something that will be scrapped when your computer arrives, just getting a head start. Getting READY:_, so to speak.

Let's shift our mental gears for a minute, and instead of thinking of the 8780 as a computer peripheral, we'll consider it in terms of being a digital sample and hold.

Our analog S/H circuits are acceptable, but they will always drift because they store information by charging a capacitor. Even if we were able to miraculously devise a capacitor with no leakage, we still have to measure the charge on the capacitor; and whatever circuit we use to do that will itself eventually drain away all the charge (I think that a Mr. Heisenberg had something to say about this, but I'm not certain). With a digital S/H, we don't have that problem, because we're storing the information as a pattern of 1's and 0's.

To use our new digital S/H we need some way to provide it with the 1's and 0's it needs to decide what voltage to produce. We need some way to "encode" our AGO keyboards.

There are lots of ways to do this, including the simple expedient shown in figure 1.

This is frequently referred to as a "brute force" encoder. When a switch closes, any diode connected to the switch line forward biases, causing a 1 to appear on the data line connected to it. The diodes are there in the first place to prevent "sneak" current paths back through the matrix. This is an acceptable encoder as long as you assume that only one key is going to be down at a time. But, when two keys are pressed

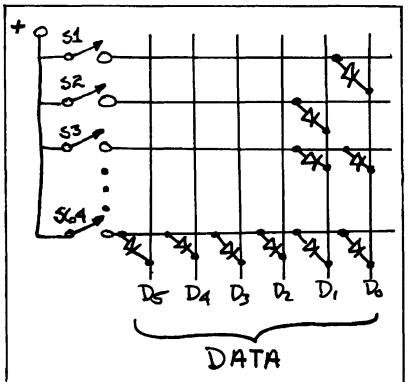


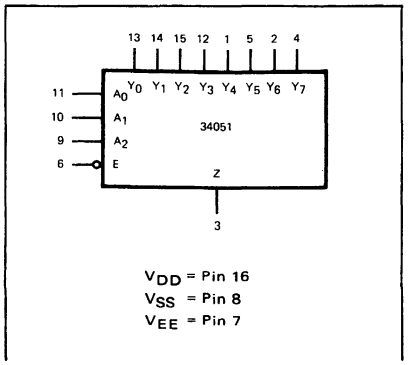
Figure 1 - Brute Force Encoder

simultaneously, the diodes act like OR gates and the data that comes out may or may not (most probably not) represent those keys. If, for example, we were to press the first two keys down at the same time, data lines D_0 and D_1 would both go high. Exactly the same situation that we had defined in figure 1 as being an indication that key 3 was down:

BUMMER

A more popular approach (because it works better) is to "scan" the keyboard a switch at a time to see if any are closed. There are LSI chips that do this with a single integrated circuit package; but, while saving design time is a great temptation, we're not going to use them. They're too expensive, and worse yet, not versatile enough to do all the things that I have in mind.

So that you can follow the design that I prefer, let me turn you on to a new part:



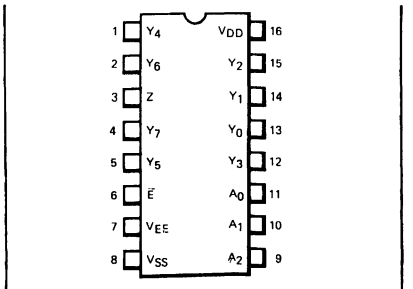


Figure 2

This is called a "4051 8 channel analog multiplexer/demultiplexer". Or, just 4051. Inside the package are 8 bilateral CMOS switches. While one side of each of these switches is tied to one of the pins Y_0 - Y_7, the other side of all the switches are commoned and connect to pin Z. In mechanical terms, it looks like this:

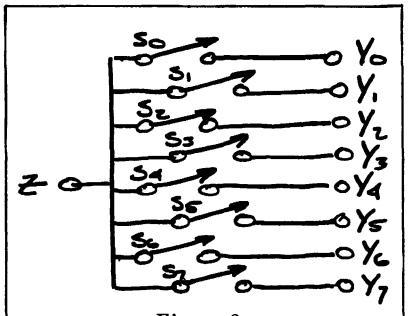


Figure 3 - Mechanical Equivalent of 4051

One of the neater things about the 4051 is that each of those switches is individually "addressable" from the pins marked A0 - A2. If I put the binary number 000 into the address pins, switch S_0 will "close". 001 causes switch S_1 to be activated, and so on to 111 which addresses S_7.

You will also notice a pin labeled E. This is an enable pin that sort of says "GO" to the rest of the circuitry in the package. As long as this pin is held at a high voltage, all of the switches will be "off", but when the E pin is grounded, the switch specified by the address currently on the A pins will close.

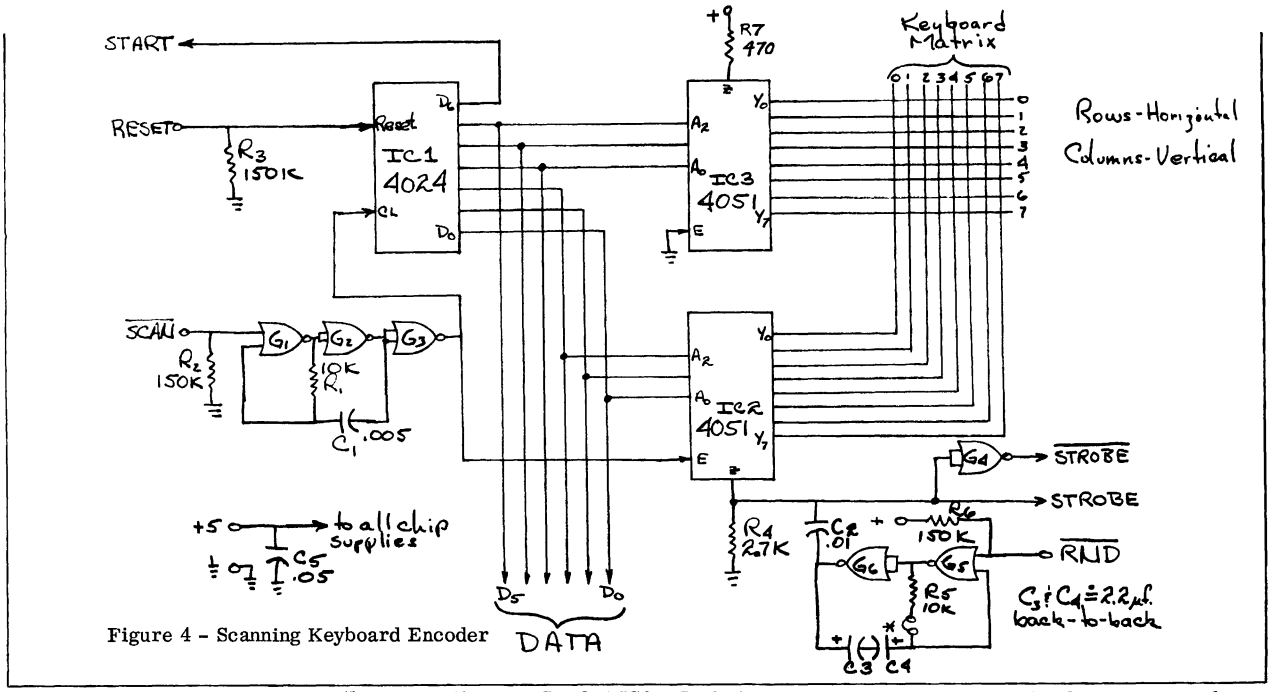


Figure 4 - Scanning Keyboard Encoder

What a terrific part. We really need to spend some time soon looking at all of the potential applications for this device. Not today, though. Today we have too many other things to do.

You're already familiar with the 4024 CMOS seven stage divider, we've used it before in other applications. Now we're going to use it again in a circuit that looks like figure 4.

This is our keyboard encoder. As far as parts go, there's not a lot to it. But it does a lot, watch.

Gates G1 and G2 along with R1 and C1 form an astable clock buffered by gate G3 that feeds the seven bit counter IC1. Notice that I can stop and start the clock by raising or lowering, respectively, the line labeled $\overline{\text{SCAN}}$. If I'm not using this line, I can simply leave it disconnected and the pull-down resistor R2 will keep the clock running.

Notice that the three LSB's from the counter (D0 - D2) are connected to the address pins of IC2 while the next three MSB's connect to the address pins on IC3 (we are going to temporarily forget about the seventh bit). Assuming that the counter starts counting at 000000, both IC2 and IC3's Z pins are connected to their Y0 pins. If these two Y0 lines are isolated from one another another nothing happens; but, if they are shorted together (by a switch at the point at which they cross in the matrix, for instance) then a current flows from the Z pin of IC3 to the Z pin of IC2 through R4 which is tied to the ground. The resulting voltage rise across R4 appears on the line labeled STROBE as a logical 1, which we can interpret as an indicator that a key is down.

When the clock cycles and the counter advances to 000001, it has no effect on IC3, but IC2's Z pin is now connected to it's Y1 pin. If those points in the matrix are isolated - nothing; if they're connected, we get a 1 on the strobe line. As you can see, each clock cycle advances the counter, which will have the effect of looking at each cross point in the matrix, one at a time. A STROBE results if the cross points are connected.

At any instant in time, the six bit number appearing on the data line is the number of the key being examined - in binary, and the status of the STROBE line will tell us whether that key is up or down.

It will also be handy at times to have a line that goes low when a down key is found, so G4 is used as an inverter to provide the complement of STROBE - $\overline{\text{STROBE}}$. (I'm tempted to say son-of-strobe, but actually NOT strobe.)

One subtle point about the 4051's that we overlooked above: the line from the clock also connects to the E pin of IC2. The effect of this is to allow a STROBE to occur only during negative half-cycles of the clock (immediately after the counter changes state) like this:

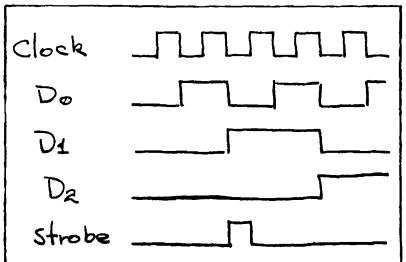


Figure 5 - Encoder Timing

which assumes that key 000010 is down. This is done for timing considerations.

Also, getting back to the counter again for a moment, we have a reset available; and while I can't think of a use for it right now, one may come up later. I bring it out on a line with a pull down resistor, R3, and label the line RESET. Raising this line to a 1 will reset the counter. Also, that seventh bit that we conveniently forgot, we can now bring out on a line labeled START. In computer application this line will serve as an indication that a scan is just starting or ending.

So, that's our all-purpose, super-gee-whiz keyboard encoder. In all of the drawings, I've shown it operating from a 5 volt supply because in computer applications we're going to be tapping power from the processor; but we're using CMOS logic here and the big reason is that it likes all different kinds of supply voltages - anywhere from 3 to 15 volts. If we retro-fit this stuff into a 4782 Road Keyboard (which as you might expect, I highly recommend) we can easily use the +9v. part of the supply that's already there to power both the encoder and the D/A.

The encoder can handle up to 64 switches (the number of cross-points in the matrix) and it will obviously work with a 5 octave keyboard. We really want to concentrate on a 37 note unit, though, since this is our standard.

No matter whose keyboards we are going to use, we are probably going to have to make some changes in the switch busses first. I'll show you on one of ours. If yours is different, I'm sure you can figure out something.

PAIA keyboards (and most others, too)

have two busses: one of which boils down to a single switch that is closed as long as any key is down. With analog S/II's, this is a signal to the circuitry to do its stuff. We don't need this anymore.

The second buss is really 37 switches, with one side of each switch tied to a common connection. We could represent it like this:

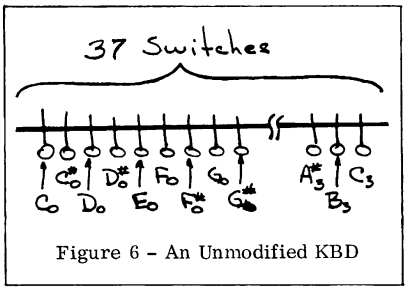


Figure 6 - An Unmodified KBD

The switch contacts that are not commoned would ordinarily go to the voltage divider board in an analog system.

We need to break these switches down into groups of 8 (giving us 4 such groups with a group of 5 keys left over) by cutting small sections (about 1/8 inch or so) out of the buss rod that runs the length of the keyboard. When you do this, don't forget that you have the keyboard upside down. Be sure that the first cut is between the first G and G# on the keyboard. I ran into structural problems after cutting the buss rod: one section of it was supported at only a single point. An easy fix for this problem was to slip short sections of clear tubing (spagetti) over the adjacent ends of the cuts, providing both insulation of the buss section and mechanical rigidity. When you're finished, what you have could be represented by figure 7.

Now we buss together the individual key switches from each group by connecting together all of the first keys in each group, all the second keys in each group, etc. Notice that again to prevent sneak current paths which could generate "phantom" keys if multiple switches were closed, we've added a diode in series with each key. When we're done, we have what's shown in figure 8.

If we now redraw what we've got and superimpose it on the matrix, we have what's shown in figure 9.

You probably noticed that the first key does not begin at note 000000, but rather picks up from row 2 of the matrix; equivalent to making it key number 010000 from the encoder's standpoint, and transposing the keyboard 16 semi-tones up-scale from the D/A's point of view.

IT DOESN'T MATTER WHERE THE FIRST KEY STARTS.

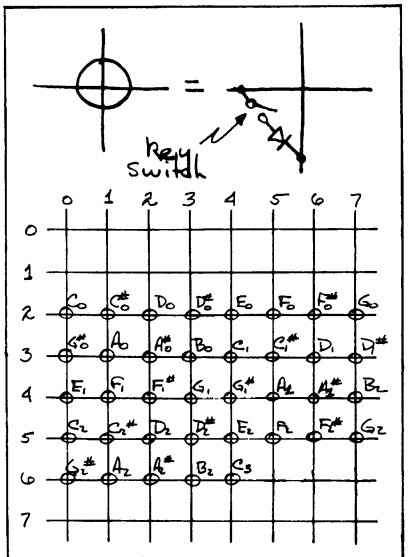


Figure 9 - The Keyboard As Matrixed

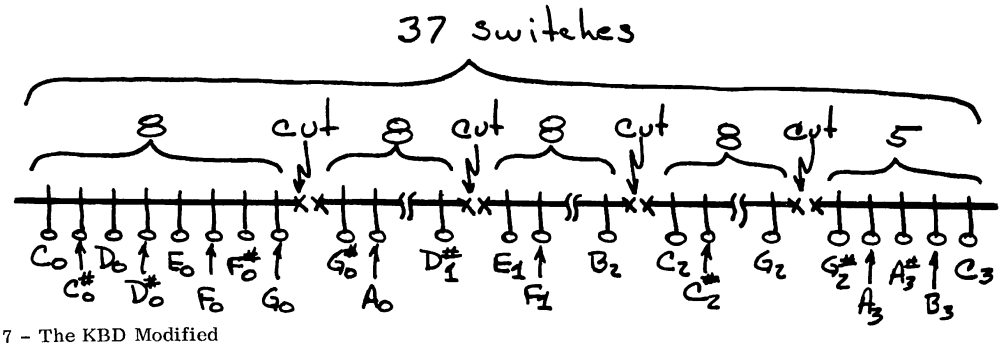


Figure 7 - The KBD Modified

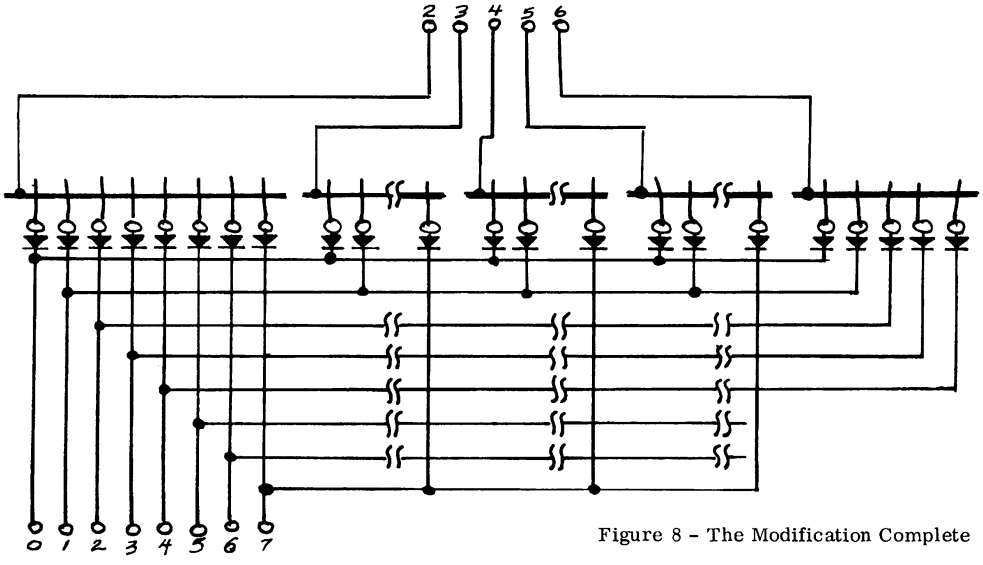


Figure 8 - The Modification Complete

Between the pitch knobs on our oscillators and the one on the D/A, we will be able to "put" the oscillator in any pitch range we want anyway.

There are a couple of good reasons for starting with key number 010000; First, I have a few computer things in mind for keys 000000 through 000111, and I want to hold them in reserve. Also, one of the things that our computer is eventually going to do for us is take care of transpositions into a new key signature, which will simply be a matter of adding to, or subtracting from, the note data the number of semi-tones by which we want to transpose. If my first key is 000000, I'm going to have a hard time transposing it down scale.

Now that I have the keyboard connected to the encoder, I'm ready to start doing things. Like replacing my old analog S/H with this shiny new digital model. There are lots of ways that I can do this. One is shown in figure 10.

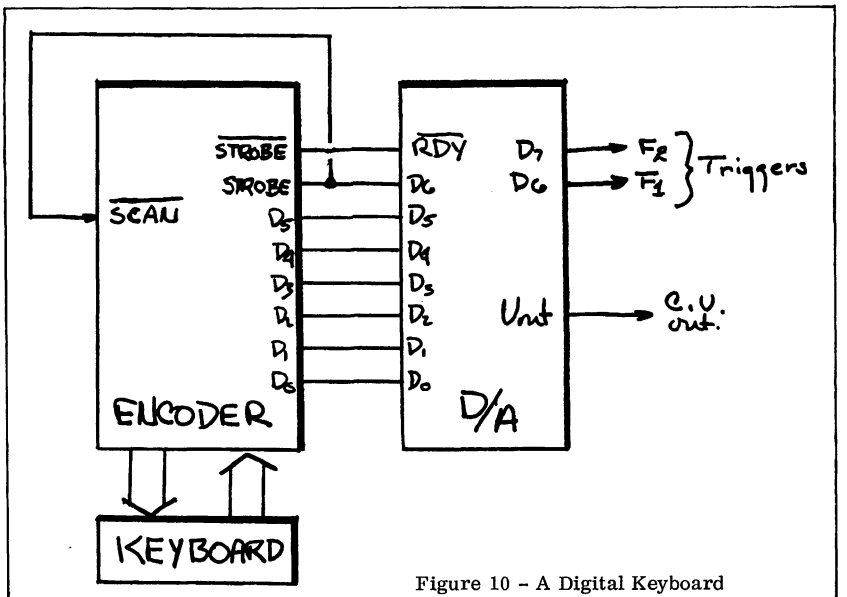


Figure 10 - A Digital Keyboard

Assuming that no keys are down, the encoder's STROBE line is at a 0 and $\overline{\text{STROBE}}$ is at a 1, making the $\overline{\text{RDY}}$ on the D/A high. The 8780's input latches are in a holding state and the activity on the data lines D_0-D_6 is invisible to the converter. This is fortunate, since the data lines are "counting" as the encoder continually looks at the keyboard.

Now, we push down a key. For the purpose of illustration let's say that it's the first key, number 010000. When the data lines next reach the state 010000, the encoder finds that the key is down, and because of that, the STROBE line goes high stopping the encoder clock, and the $\overline{\text{STROBE}}$ line goes low which takes the D/A's $\overline{\text{RDY}}$ line to a 0 putting the D/A's latches in a pass state. The new note data (010000) is strobed into the converter and a control voltage representing that key appears at the control voltage output of the D/A. The STROBE line from the encoder also connects to the D_6 input of the D/A, which appears at the D/A output panel as the first trigger flag (F1), so we have a trigger showing that a key is down. And this trigger is used the same way we would a trigger from the analog system.

As long as the key is down, the system is going to sit in this configuration. But, when I release the key, new things happen. Almost simultaneously STROBE goes low which removes the trigger flag D6 (indicating that the key is now up) and allows the clock to start again (searching for the next key down). Simultaneously, $\overline{\text{STROBE}}$ goes high forcing the $\overline{\text{RDY}}$ line on the D/A high which puts the latched in a holding condition - and what they're holding is data on the last key that was down.

This is behaving exactly like the old analog system that we had, except, as I already mentioned, it doesn't drift. AND it gets rid of that annoying "in between" note that we had with the old keyboard if two notes were pressed at the same time (since the clock stops, the encoder can "see" only one down key at a time). AND, it doesn't have 37 adjustments to tune it; now there are none.

Let me show you something that this keyboard can do that our others can't.

Suppose that we remove the wire connection between the encoder's STROBE output and $\overline{\text{SCAN}}$ input. You will remember that this was the thing that caused the clock to stop when a key was found down. If we replace the wire with a capacitor, say about .22 mfd. or so, we have generated a little time delay in this loop. The clock will stop when a key is found down, but only temporarily - until the capacitor discharges - then it is going to go looking for the next key down. If, in the process of searching, the encoder finds another key down, it will strobe it into the latches, hold for the time delay, and then go searching again. With this arrangement, if two keys are held down, the output of the D/A will alternate between the two, and what we will hear is a trilling between these two notes. If three keys are held down, each note will be heard in turn and while this is not polytonic by any stretch of the imagination, it can certainly sound that way.

Can you imagine what the effect of pushing down a large number of keys will be? I call it the "orgasmatronic glide" but everyone here thinks that's a terrible name.

Anyway, the arpeggiation gimmick is slick and if you wish it can be left in place and bypassed with a switch when not used as shown in figure 11.

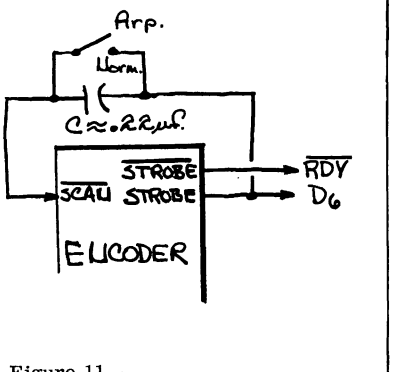


Figure 11 -
The "arpeggiation" Modification

Here's another one.

You may have noticed that there is an input to the encoder that I hadn't mentioned; the one labeled (innocuously) $\overline{\text{RND}}$. This line is normally held high by R5, but when pulled low momentarily it causes G5 and G6 to both change state which in turn activates the strobe line - even though there are no keys down.

The effect of this is that whenever we activate this line, whatever number happens to be on the data buss at that instant will be strobed into the D/A. Since the encoder clock is working very fast, there is no way to know in advance what the number on the data lines will be. As you've probably guessed, RND stands for RANDOM, since that is the effect of this input. It causes a random note to be strobed into the D/A.

If it occurs to you that there is a lot of activity around G5 and G6, what with R5, R6, C3 and C4 and the funny little jumper marked (*) being there, you're right. This circuit not only buffers the RND input line, it is also a slow clock. If we hold the RND down for more than just an instant, square waves begin appearing at the output of G6. And, naturally, for each cycle of the output of G6 a new random note is strobed into the D/A. With the values shown, the tempo of this clock is several cycles per second. That's a bunch, and that's where the (*) marked jumper in series with R5 comes in. By replacing this jumper with a pot (about 500K is a good value) we've picked up a control of the tempo of this circuit.

By adjusting this new tempo control we can effect not only the rate at which random notes are thrown out, but also the character of the notes (whether they appear to be really random, or run upscale, or downscale, or whatever).

To understand why the character of the notes is effected, imagine for a moment that the tempo of the RANDOM clock is exactly 1/64 that of the scanning clock. Under these conditions, the RANDOM clock is going to pull one note from the encoder for each complete encoder scan. Since 64 notes constitutes an entire scan, the RANDOM note that we pull will not be random at all. It will be the same note each time.

Suppose, now, that the RANDOM clock is running at exactly 1/65 the tempo of the scan clock. Now each time the RANDOM clock says "sample", the scanner will have gone through a complet cycle plus one note. Each succeeding sample will pull a note that is one semi-tone higher in pitch than the previous sample, and we will hear an ascending series of semi-tones that increments by one semi-tone for each event.

If the RANDOM clock is running at 1/63 the frequency of the scan clock we will have a similar situation except that the note pulled each time will be one semi-tone lower in pitch than the preceeding semi-tone.

Actually, for any practical situation, the RANDOM clock is going to be running several hundred or thousand times slower than the scanning clock; but the principle still applies. Small changes in the RANDOM clock rate will produce wide variations in the character and organization of the notes that are "randomly" pulled from the encoder.

Out of space and out of time, again. And so much left to do. It will have to wait for next time.

Speaking of next time - here are some things that we're going to do:

We're going to look at a memory add-on for the encoder, D/A combination that will allow you to do some terrific digital sequencer things. We're also going to look at an expansion system that will convert what we've done so far into a polytonic (phonic) keyboard. Also we'll have a story on a touch keyboard - the easy way, and will look at ways that this kind of thing can be tied into our encoder, D/A set-up.

And, I think, our computer will be ready. We've put a lot of time into configuring it for maximum usefulness either as a stand-alone micro-processor trainer or for use with the music stuff. I believe that the time has been well spent. When you see all of the things that this system will do for you it's going to:

BLOW YOU AWAY

No kidding.

POST SCRIPTS

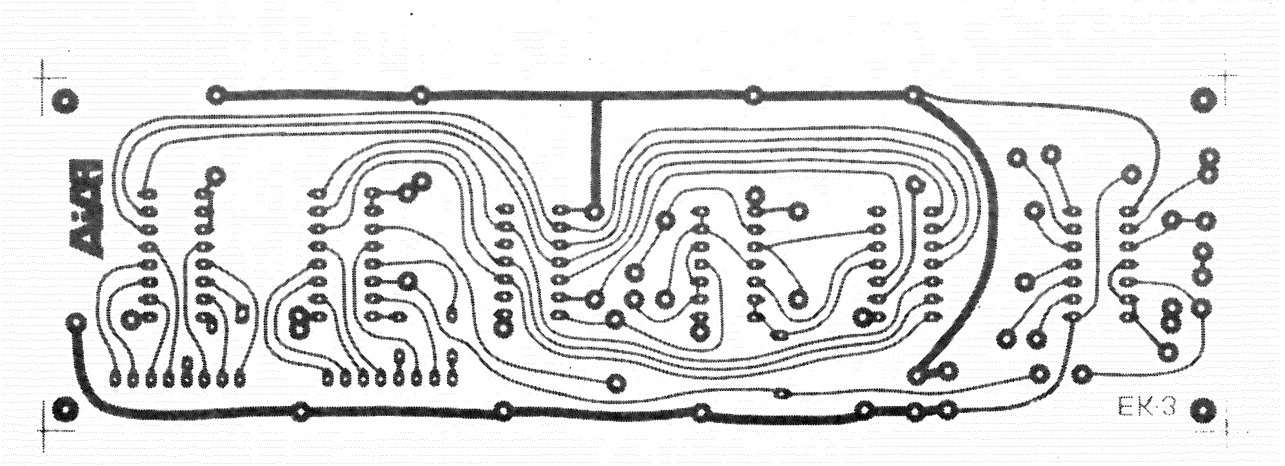
NOTE:

The PAIA Experimenter's Kit series is not intended for the novice builder. They are intended to provide the experimenter with a place to start on what will hopefully be a series of interesting and enlightening projects at the very lowest possible cost.

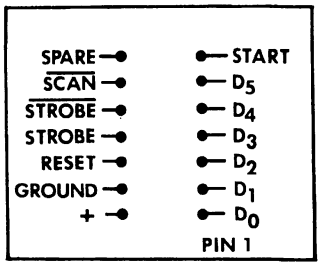
Because of this, parts that are considered to be either optional or easily obtainable from other sources (your "junk box" for instance) are not included. Also, circuit boards in this series are not normally printed with parts placement designations and assembly instructions are minimal, with most of the narrative type textual material concentrating on "how it works" and "how to make it do other things." If you feel that this approach does not serve your purposes you should return this item immediately for a refund.

Parts placement for the EK-3 circuit board is shown below.

Note that with the exception of the RND input, all input and output lines come together at the 14 pin DIP configuration between IC4 and IC5 on the circuit board. A DIP socket and connector may be used here if desired for easy connection and disconnection (a nice touch, but not highly recommended).



Below is an enlarged version of this I/O cluster which may be cut out and placed in the vicinity of the EK-3 for easy reference.



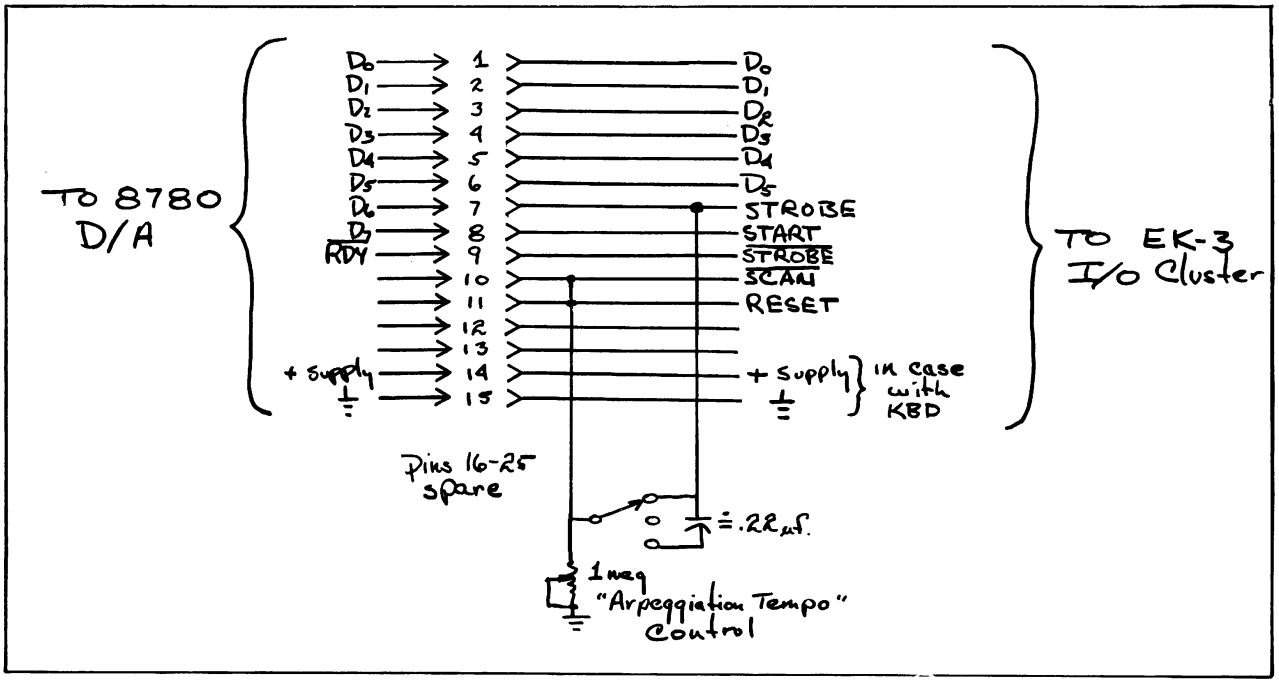
The "standard" connectors that we will be using for the complete kit version of these devices will be 25 pin "DB25" type sockets and plugs. If you decide to use these connectors, we recommend that they be wired as listed below:

PIN #	8780 D/A (plug)	EK-3 encoder (socket)
1	D_0	D_0
2	D_1	D_1
3	D_2	D_2
4	D_3	D_3
5	D_4	D_4
6	D_5	D_5
7	D_6	STROBE
8	D_7	START
9	RDY	$\overline{\text{STROBE}}$
10	N/C	$\overline{\text{SCAN}}$
11	N/C	RESET
12	N/C	N/C
13	N/C	N/C
14	+ supply	+supply
15	(⏚)	(⏚)

Some forethought has gone into the configuration of these connectors with additional scheduled elements of the series in mind.

For example, if this scheme is followed, the arpeggiation gimmick described in the text would be added as shown in the figure below.

The three position switch can then select a mode of operation in which the clock stops when a down key is found (up position); a mode in which the clock does not stop when a down key is found, and this mode will be used in polytonic retro-fits (middle position); and the arpeggiation mode in which the clock stops momentarily for down keys (bottom position). As is indicated, a control of the arpeggiation rate (within limits) may be added with the 1 meg potentiometer shown.



LAB NOTES

IN PURSUIT OF THE WILD QuASH

by John S. Simonton, Jr.

Now that we have a way to interface our synthesizers to computers - the 8780 D/A - we can begin thinking of ways to independently control large numbers of musical elements simultaneously. Lots of VCOs, lots of VCFs.

The first time that you think of this your reaction may be something like:

WOW! - ALL THOSE D/As.

Multiple D/As (one for each control "channel") would be a possible way to go. An expensive way - at $35.00 each, controlling just 4 VCOs means almost $150 worth of just D/As.

There's a much cheaper way.

You may find this a little circuitous, after working so hard at our digital interface, but we're going back to analog Sample and Hold circuits.

Now wait, don't panic. These S/H's are nothing like the ones that we're accustomed to. They don't have to hold a voltage steady for a long period of time - only a few milli-seconds. Long before even that short time has passed we will have used the computer to come back and re-write the correct voltage into the circuit. Computer re-freshed S/Hs.

Magic!

When you're designing a S/H to be good for only a fractional part of a second it gets really easy. Like this:

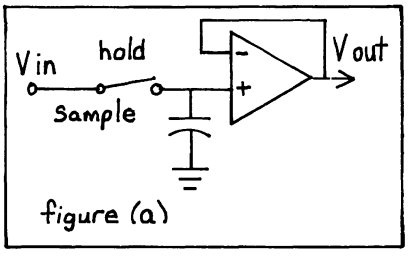


figure (a)

I'm sure that we've all seen this kind of thing before. It's an op-amp used as a unity gain voltage follower.

When it comes time to take a sample, the switch closes causing the capacitor to charge up to the input voltage. The output of the voltage follower "follows" this voltage (what else?), and when the switch opens again, the capacitor "remembers" the voltage.

One of the characteristics of this circuit is that the "+" input represents a very high input impedance to any load that it sees. A relatively small capacitor can accurately hold a voltage for almost a second.

Now, we're not going to use a mechanical switch here. Last time, we looked at the 4051 multiplexer and decided that we would be using it a lot. And we are, just not this time.

This time, we're going to use a very close relative of the 4051 - the 4052 (I defy you to get any closer than that). The 4052 looks like this:

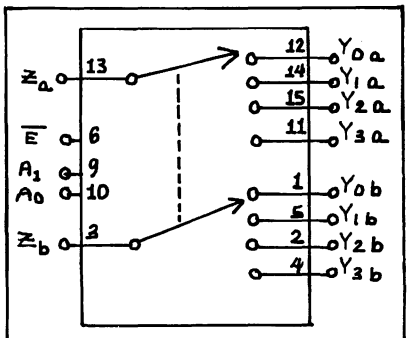


figure (b)

and whereas the 4051 was an electronic equivalent of a Single Pole Eight Throw switch, the 4052 is like a Double Pole Four Throw one.

Which pairs of switches are to close is specified by the two address lines (A_0 & A_1). The switches actually close when the $\overline{E}$ pin goes to ground.

Using 1/2 of one of these devices we can come up with a Quad Addressable Sample and Hold (QuASH?) that looks like figure C, and it works about the way that it looks. An address applied to the A_0 and A_1 pins sets up one of the four switches and when the $\overline{E}$ pin is taken to ground that switch closes connecting the output of the D/A to the selected S/H. Simple.

That takes care of our control voltage output - but there are still other things to think about. For instance, we need a trigger flag (gate signal) to go along with each of the control voltages to take care of things like triggering envelope generators. *(1)

An easy way to handle this is to use the other 1/2 of the 4052 to route one of the two trigger flags available from the D/A to an output corresponding to the control voltage output. And since we're time sharing the D/A we also need some way to hold the status of that flag during the times that other control

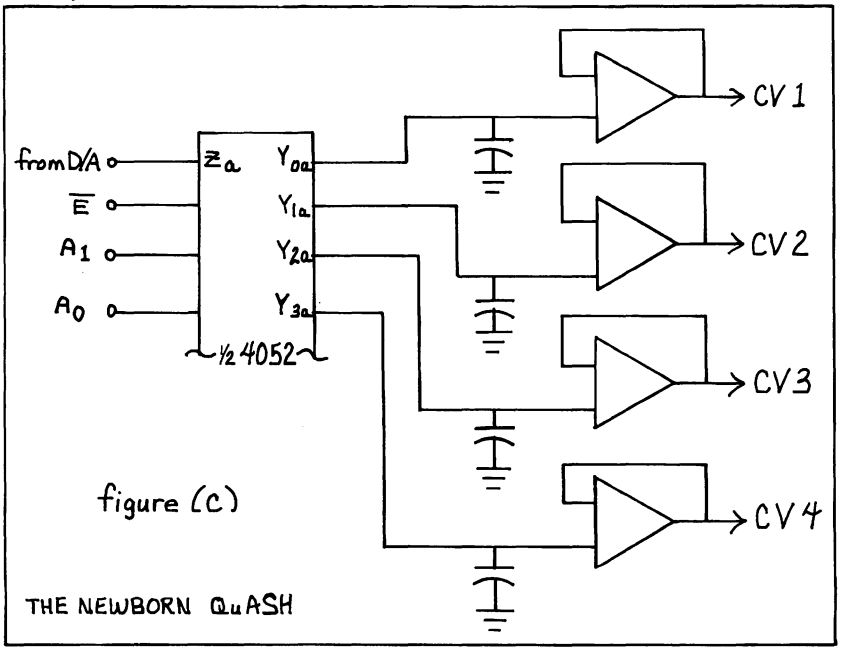


figure (c)

THE NEWBORN QuASH

channels are being addressed. Do latches come to mind? Forget them - in this application they're going to be far too expensive and complex by the time we get them to act the way we want.

Instead, we'll use a small capacitor and a CMOS inverter like this:

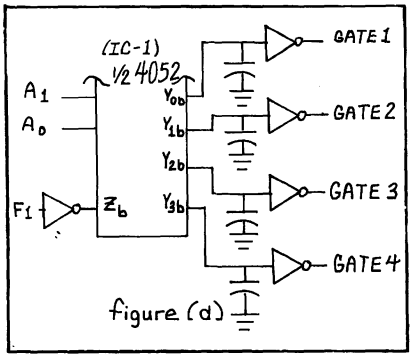


figure (d)

This is a little S/H in its own right - but it doesn't hold an analog voltage, only a "1" (output high) or "0" (output low).

Oh, yes - since we are buffering the condition of the capacitor with an inverter, we need to also invert the trigger line going into the 4052 so that everything comes out right. That's why that other inverter goes between the trigger flag line from the D/A (F_1) and the Z pin of the 4052.

But, there are two trigger flags available from the D/A - and here we a are only using one of them. Waste, ugh.

Let's do something neat with the left over flag, something really sexy. Let's use it to:

SELECT GLIDE

(tah-dah)

You may think that because we're time sharing the D/A we've eliminated the possibility of doing things like this, but we haven't. In a functioning system the S/H's are being up-dated so fast that we can in fact generate glide the same way that we did in our old pure analog system, simply by placing a variable resistor in front of the holding capacitor. We'll use a regular 4066 Quad Bilateral Switch to turn the glide off by shorting out the resistor (so that the glide is on when the switch is off), and to latch the status of this glide bit we'll use the same capacitor/inverter trick that we used on the other flag. One section of this circuitry looks like figure e.

For programming reasons, it will be handy to have the glide select bit (which is now flag 2) be a "1" when the glide is enabled and that requires a second inversion - between the trigger output of the D/A and the Z pin of this new 4052.

And now here we are with 1/2 of a 4052 left over.

Don't you believe it.

Since we will frequently have more than a single synthesizer module controlled from one of our control voltage outputs (two VCO's or a VCO and VCF would be two typical cases), it will be handy to have a modulation input associated with each control channel so that all modules driven from that channel will experience the modulation at the same time.

Another thing that ties into this is that our D/a is an exponential converter of sorts and so for the first time gives us the opportunity to do equally tempered vibrato (for example) with our linear oscillators.

We'll use the left over section of 4052 to multiplex a modulation voltage back into the D/A in the same way that we multiplexed the control voltage out. Like this:

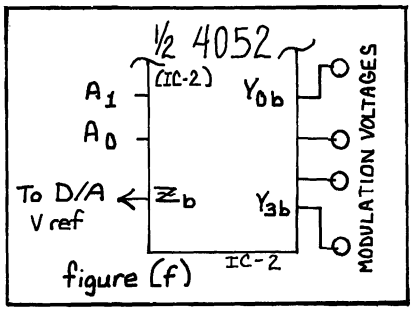


figure (f)

Because the modulation voltage corresponding to a given control channel is applied to the D/A only when that channel is re-freshed, you may think you will be able to hear the modulating influence as a series of steps. But you don't for the same reason that the glide doesn't appear to be a series of steps. Everything is just happening too fast.

One last detail and we're done with the design of this circuit.

Addressing (selecting) one S/H out of the four on this card is of course handled by the address pins of the 4052's. But, many systems will not stop at just 4 outputs; some folks, I'm sure, will want to take the system to the limit (in practical terms about 32 outputs) - which implies that more than one of these cards may (and probably will) be used in a system. We need a way to be able to select not only one of the four outputs on this card, but also a means of selecting one card from many.

Here's the address decoding scheme we'll be using:

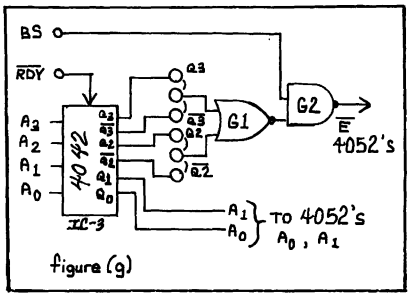


figure (g)

The 4042 Quad Latch is an old friend - here we' re using it to latch the computer's 4 least significant address bits at the same time that data is put out to the D/A (the RDY line on this card is connected the same as the corresponding line on the D/A).

We want to latch these address lines because the WRITE cycle of any computer we come up with is going to be much shorter than the time required for settling of the D/A and S/Hs. Latching the address lines allows us to output

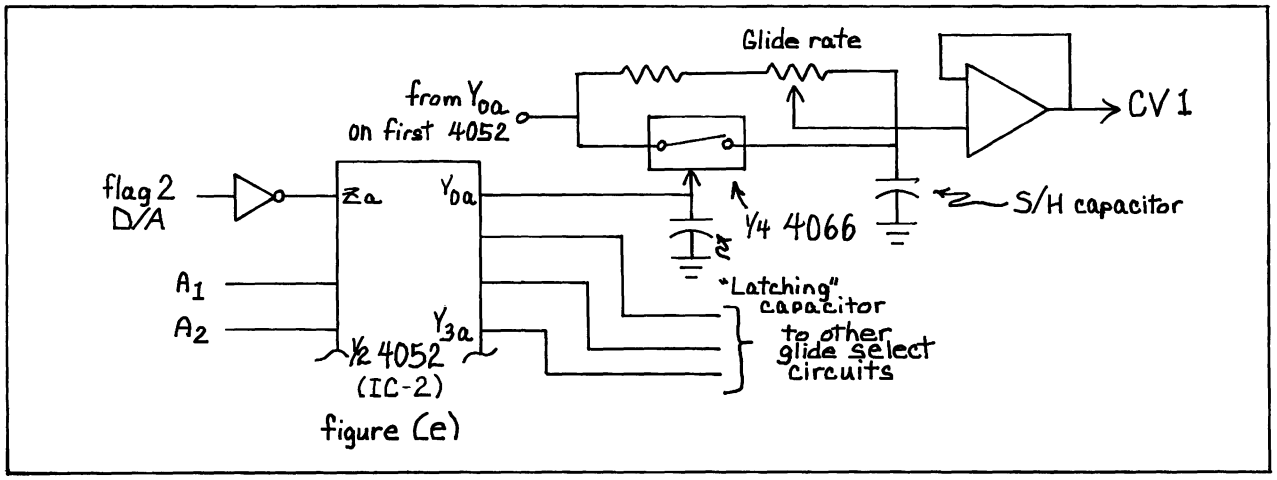


figure (e)

data and then wait (or do something else) while these analog circuits get to where they're supposed to be. *(2)

Notice that the Q_0 and Q_1 outputs of the latch - corresponding to the two least significant address bits - go directly to the 4052's where they serve to select one of the four outputs.

Notice also that Q_2 and its complement $\overline{Q_2}$ as well as Q_3 and $\overline{Q_3}$ from the 4042 come out to pads on the circuit board. By jumpering these outputs to the inputs of the NOR gate G1 we can determine which group of addresses the card we're working with represents.

For example, if we connect the inputs of G1 to $\overline{Q_2}$ and $\overline{Q_3}$ then this block of four S/H's occupies the addresses 00XX in binary where XX represents the bits that select one of the four S/H. Address 0000 corresponds to the first S/H, 0001 to the second, and so on. By connecting the inputs of G1 to Q_2 and $\overline{Q_3}$, the S/H's occupy the address 01XX. The first S/H is 0100, the second 0101, and like that. This scheme allows us to easily use up to four of these expanders (16 outputs) in a system without needing to do anything but set the jumpers properly.

You will notice that there is another line coming out of this decoding circuit which is labeled "BS". This is not my opinion of this whole mess, it's a means by which we may expand the system beyond even four expander modules - BS stands for "Bank Select" and as long as this line is held at a logical "1" level the system operates as described to this point.

But, when the BS line is pulled low one input of the NAND gate G2 is not fulfilled resulting in its output being high which in turn holds the 4052's enabling input $\overline{(E)}$ high - which means that none of the switches in the multiplexer will close (even if addressed otherwise) and none of the S/H's will be selected.

External decoding circuitry is required to drive the BS input, naturally, but we would begin to need external circuitry at about this point anyway to buffer address lines. The decoding required here will be covered in the instruction manual for this kit.

When we tie all of these bits and pieces together, we come up with a thing that looks like figure H, our complete QuASH. And in the interest of saving space and time, we will from this point forward represent it with the symbol shown (at least until we can come up with something more abbreviated). The knobs in the output "boxes", by the way, represent the glide rate controls associated with each output channel.

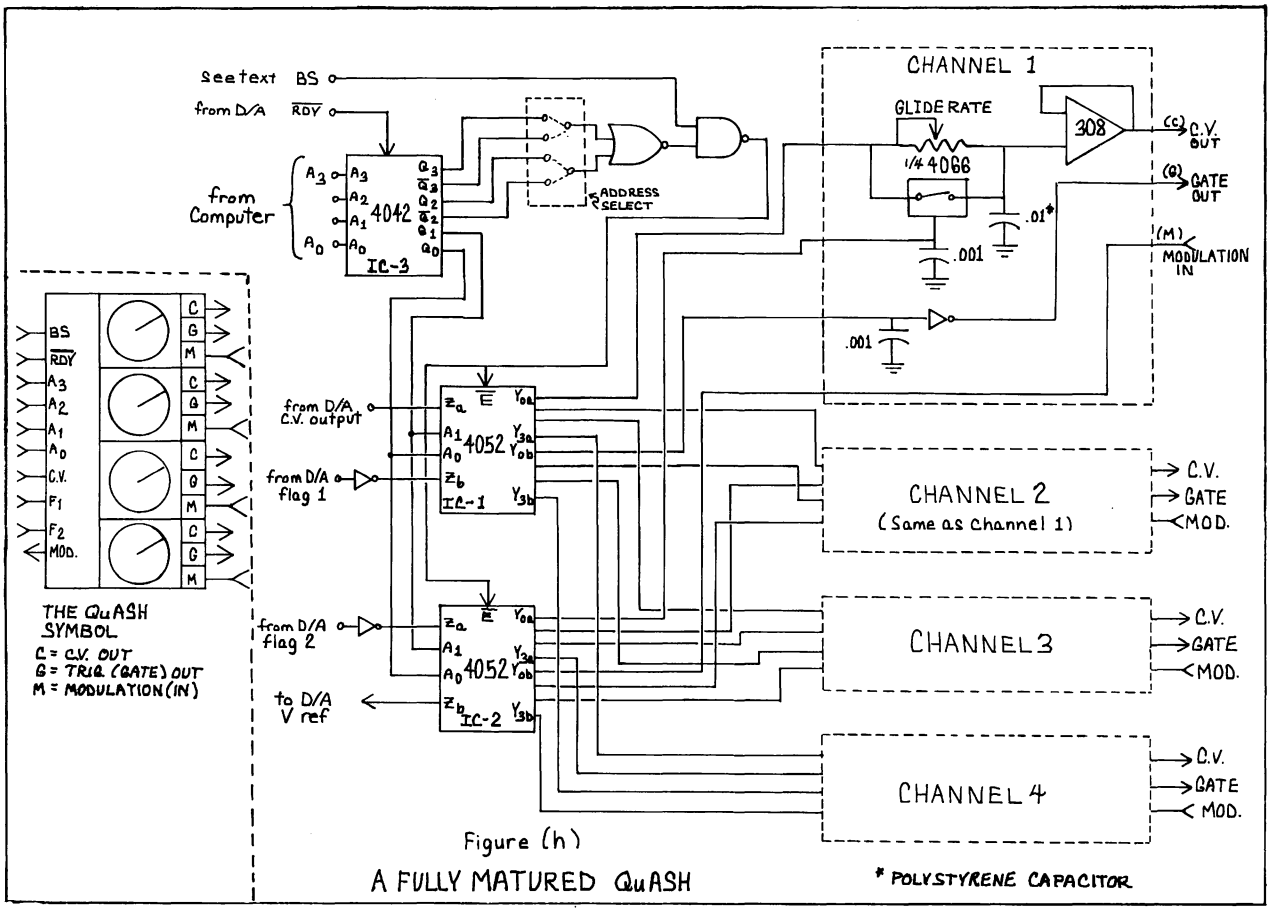


Figure (h)

A FULLY MATURED QuASH

* POLYSTYRENE CAPACITOR

NOTES:

*(1) Those of you who have been thinking about this stuff for a while will, of course, recognize the imminent demise of the ADSR. Providing Attack, Decay, Sustain and Release parameters is one of the easier tasks to turn over to the computer entirely. On the other hand, I've played with this some and can testify that varying the position of a knob is handier - in this case - than changing parameters in the memory of the machine. Some Hardware ADSRs mixed with some Software ADSRs seems a good compromise.

*(2) This off-hand statement is not meant to imply a wait in human terms (major fractions of a second), but rather a wait in machine terms - micro-seconds. You don't have to wait for a GLIDE to finish (for instance) before doing something else.

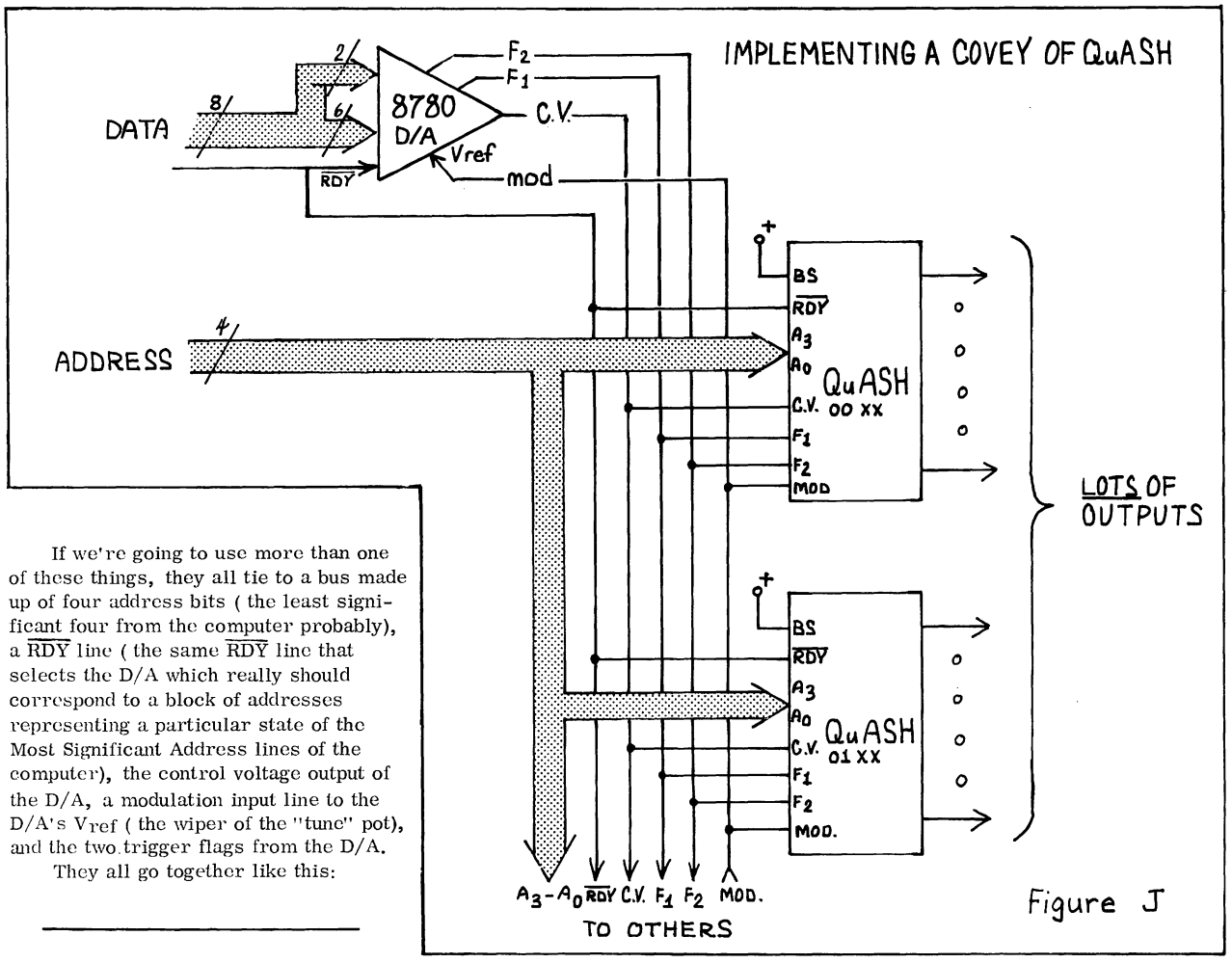


Figure J

If we're going to use more than one of these things, they all tie to a bus made up of four address bits (the least significant four from the computer probably), a $\overline{\text{RDY}}$ line (the same $\overline{\text{RDY}}$ line that selects the D/A which really should correspond to a block of addresses representing a particular state of the Most Significant Address lines of the computer), the control voltage output of the D/A, a modulation input line to the D/A's Vref (the wiper of the "tune" pot), and the two trigger flags from the D/A.

They all go together like this:

THE POLYPHONIC SYNTHESIZER

LAB NOTES

By
John S. Simonton, Jr.

We've come a long way over the last year in terms of developing a series of digitally interfaced modules that will allow computer control of music synthesizers. I suppose that the time has come to look at tieing them all together, with the computer, and begin doing interesting things.

I had wanted to start with "the ultimate sequencer programs" but am not completely happy with them yet. They still need a little polishing.

Instead, we'll start with what should be another popular system:

THE POLYPHONIC SYNTHESIZER

Which is a much simpler job than the ultimate sequencer.

I would like to go through the system showing specific ways to do things for a variety of manufacturers equipment but that just isn't practical. Instead, we'll look at a completely PAIA based system and assume that if you are using different equipment you are familiar enough with it to make whatever changes are necessary.

Oh, one more thing before we begin, be sure that you understand that there are a wide variety of ways to do polyphonic synthesizers. This is only one of them. I hope that the algorhythm used here works for you. It's one of many, some with sort of special quirks that make them useful in certain situations but difficult to work with generally - This seems to be good general purpose way. Ready? We have lots to do and little space and time; here we go.

THE HARDWARE

Most of the hardware that we'll be using has been described here over the last year (or so). For the controller portion of this system we'll need:

1) AN ENCODED KEYBOARD
   8782 or EK-3 retro-fitted equivalent
2) A COMPUTER
   An 8700 in it's minimum configuration will run the programs that we'll list. A cassette interface system is useful to the point of being almost mandatory. We'll show some new panels and stuff to make it all pretty.
3) DIGITAL/ANALOG CONVERTER AND SAMPLE AND HOLDS
   the 8780/8781 system.

And, of course, we'll also need as much synthesizer as we think is necessary.

With all of the items listed, various wiring schedules have been mentioned for doing various non-computer things. We now need to establish some standards for this new use, a computer based polyphonic system.

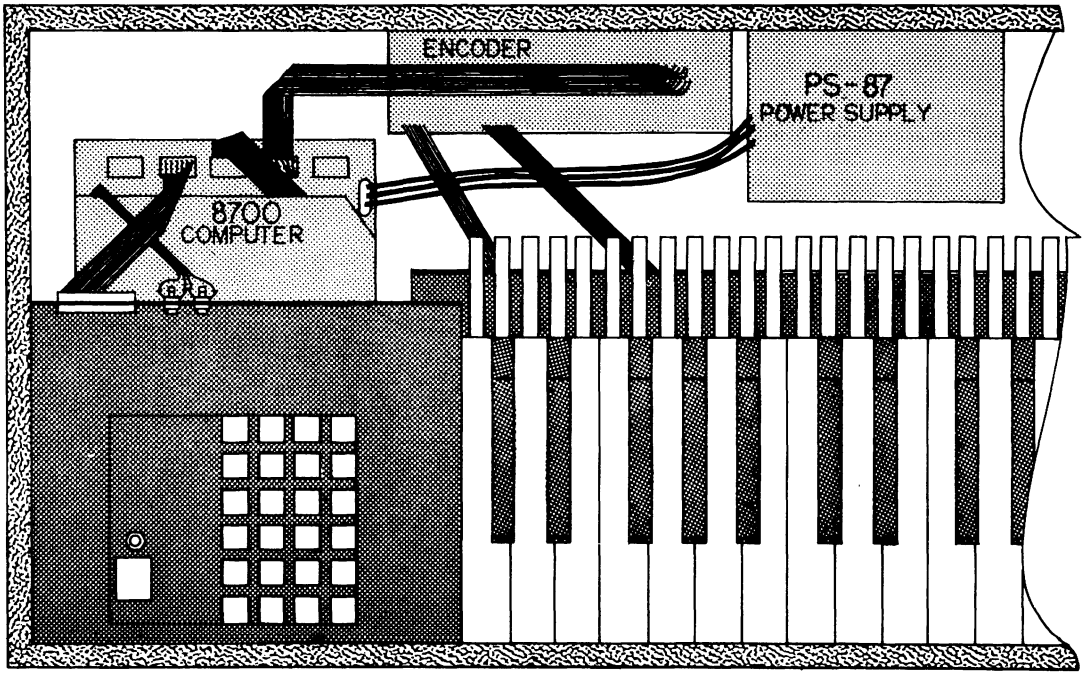
If we choose wisely, we should come up with a standard that has plenty of room for future growth. Some consideration has gone into the system which follows and I believe that it will serve our needs for some time to come.

Many of you will already have much of this wiring done, as much of it is simply an extension of what we've done before. Check carefully to be sure your wiring is to this new standard.

THE KEYBOARD

Let's go ahead and configure this system from the beginning so that the computer fits in the synthesizer cases that we've been using. All of the parts will fit in the case like this:

figure 1. computer/synthesizer sub-module placement.



PAIA 8700 COMPUTER, POWER SUPPLY AND KEYBOARD ENCODER RETRO-FIT TO 4700 OR 8700 SERIES KEYBOARD.

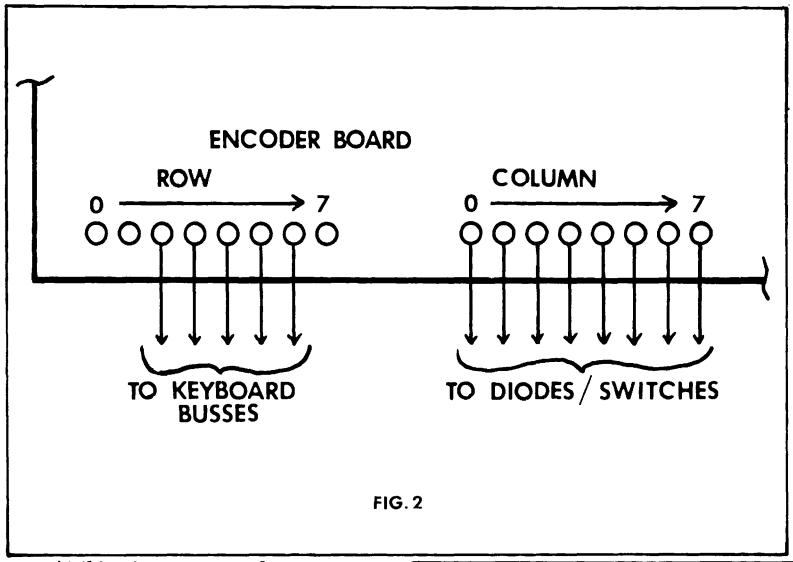


FIG. 2

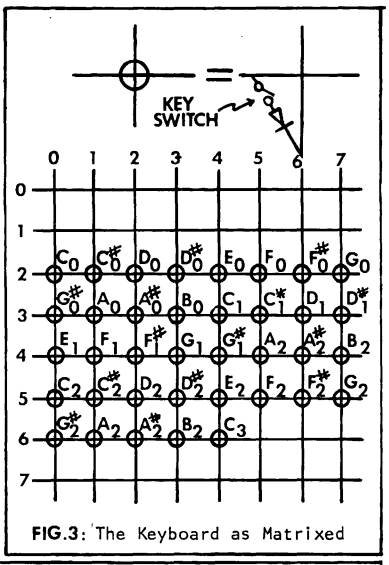


FIG.3: The Keyboard as Matrixed

At this stage you may have more dis-assembly to do than assembly. Particularly, the old control panel of the keyboard is removed to make room for the computer and any unregulated supply that was powering your keyboard encoder is replaced with a PS-87 which supplies all digital power for the entire system. This is going to give you a few parts for your "bench stock", the old power supply components and a couple of push-buttons, but some of the parts we will be re-using. Don't throw anything away.

KEYBOARD TO ENCODER CONNECTIONS

Maximum useability of the system would seem at first to depend on where the AGO keyboard switches appear in the key matrix. We want them in the middle so that we have as much room to transpose down in pitch as we do for up-scale transpositions. Some 8782 instructions had the keyboard placed 8 switch positions below where it should be for this ideal. The "column" connections are fine, but the "row" connections on these keyboards will need to be "slid up one" so that they conform to the configuration as shown in figure 2.

This will place the keyboard more or less in the middle of the matrix as shown in figure 3. This is really a fine point, and the system will work OK in most applications almost no matter where in the matrix the keys are, but go ahead and change now so that you won't be limited in the future.

ENCODER MODIFICATIONS

We don't need any of the "trick" things that we used when we didn't have a computer (the orgasmatronic glide circuit, etc.), just the bare-bones encoder. You may remove all push-buttons slide switches, pots etc.; most of these will come out when you remove the old front panel.

ENCODER TO COMPUTER

If your system previously had a DB-25 female connector tied to the output of the encoder, desolder it (carefully - whistling may make the job seem easier). In place of the DB-25 connector, we now need to terminate the output of the encoder in a DIP header that will mate with the INPUT PORT #1 (J4) connector on the rear edge of the 8700 computer board.

These connections should be made as in figure 4.

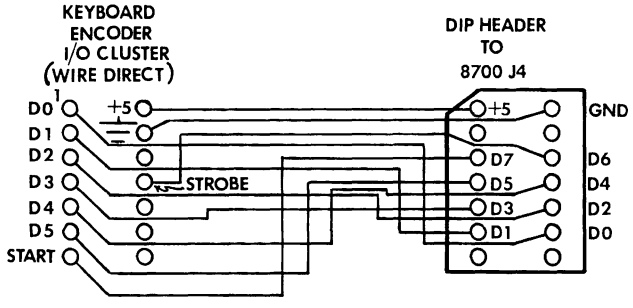


FIG. 4

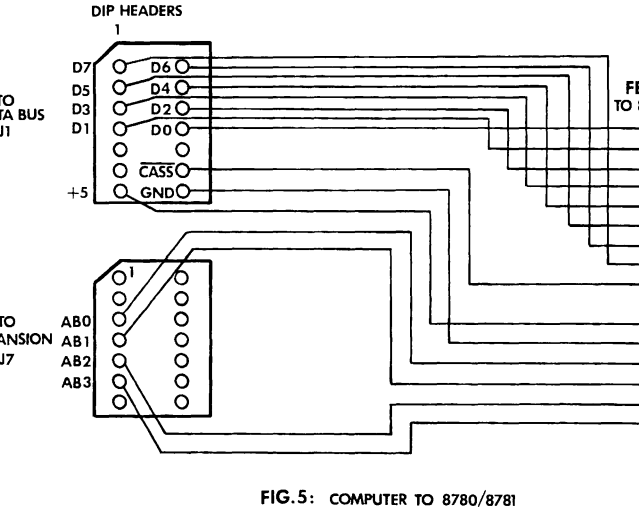


FIG. 5: COMPUTER TO 8780/8781

These connections should also be made carefully and the DIP header pins well heat-sinked to prevent melting the plastic header. NOTE that while many of the non-computer applications used the STROBE line to trigger the D/A, here we ignore this line and instead use the STROBE as the seventh data bit (D6) of the interface.

Similarly, the encoder's START line becomes the 8th data bit (D7).

Also, you will notice that power to

the encoder is picked up through this connection from the 8700 itself.

COMPUTER TO SYNTHESIZER HEAD

So that our resulting system can be easily broken down into two separate units (computer/keyboard and synthesizer head), this is the place to use the DB-25 connector that was salvaged from the old keyboard front panel.

Connections should be made between the female DB-25 connector and a pair of DIP headers like those in figure 5.

NOTE that the first header (P2) provides data lines and the $\overline{\text{CASS}}$ select signal (our 8780/8781 shares this output structure) while the second connector (P3) provides the address lines required by the QuASH.

8780/8781 WIRING

The male DB-25 connector that terminates the cable to the 8781 is wired in what is essentially an expanded version of our previous standard so that here you are faced more with adding wires than re-arranging them.

Connect these elements together as in figure 6.

This wiring schedule is examined in detail in the 8781 QuASH assembly manual. An important thing to notice here is the way the grounds are handled. Note that the ⏚ (ground) pin on the rear of the 8780 board serves as the central ground for both analog (synthesizer) and digital power distribution. This grounding scheme is important to prevent ground loop problems and should be followed exactly. This entire 8780/8781 assembly should be mounted in the synthesizer head cabinet.

FINAL ASSEMBLY

Finally, make arrangements for physically mounting the computer in the keyboard case by first mounting the computer to a suitable front panel as shown. (See figure 7)*

And don't forget to provide a socket at the 8700's expansion connector (J7) or to mate P3 with this socket before assembling the computer/front panel. If the cassette interface is being used, terminate the input and output lines in miniature phone jacks as shown in figure 8.

Plug all the connectors together and you should be ready to load a program.

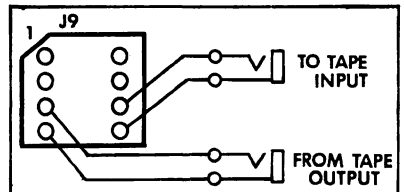


FIG.8: CASSETTE CONNECTIONS

THE PROGRAM

The polyphonic program that we'll be using is called simply:

POLY 1.0

This program supports up to 8 output channels the way that it is written and can be easily modified to provide for more.

POLY 1.0 allocates synthesizer resources to keyboard requirements using this algorhthm:

1) Output all notes appearing in the output buffer area (NTABLE) after adding the corresponding transposing figure from TTABLE. Go to 2.
2) Wait for keyboard scan to start and place a list of all keys currently being held down in the input buffer area (KTABLE). When buffer full or scan complete go to 3.

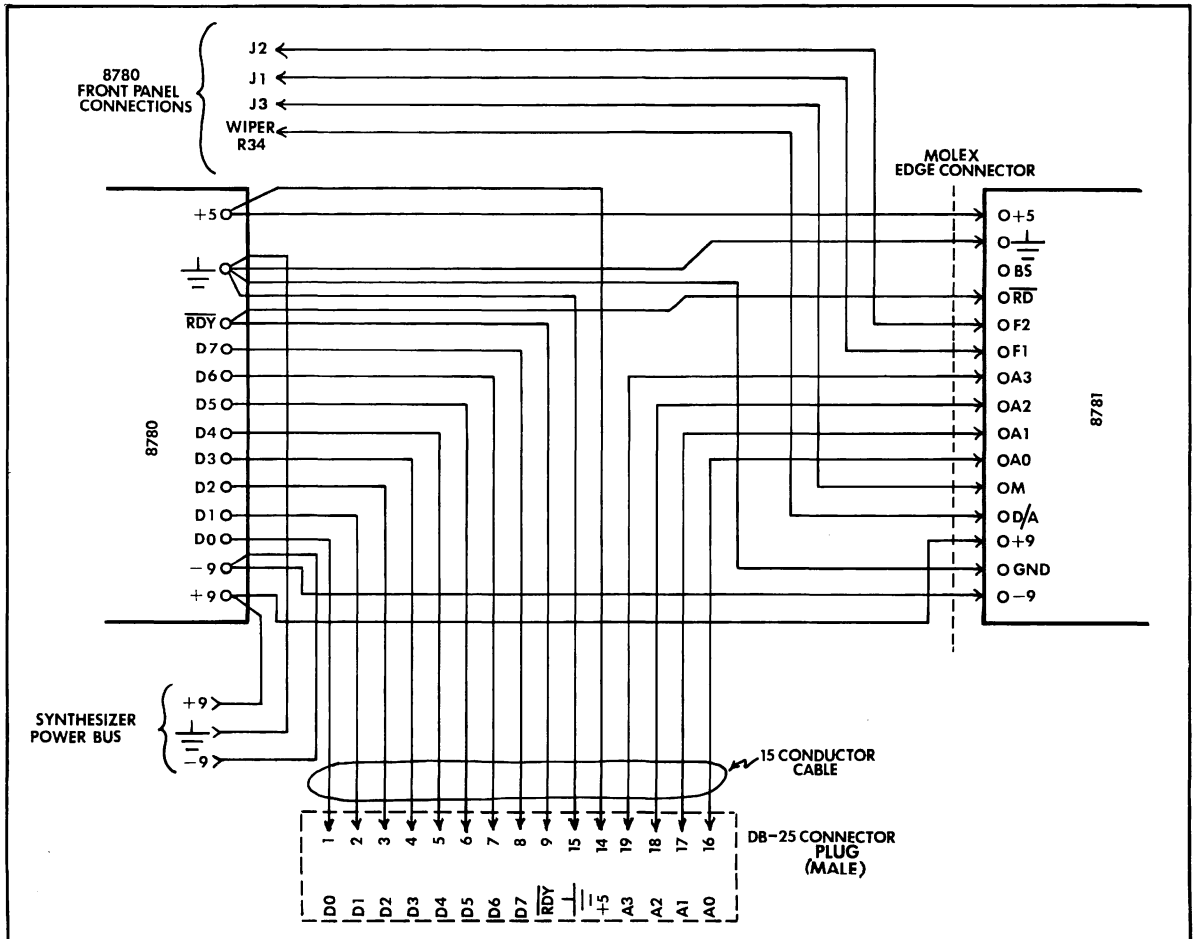
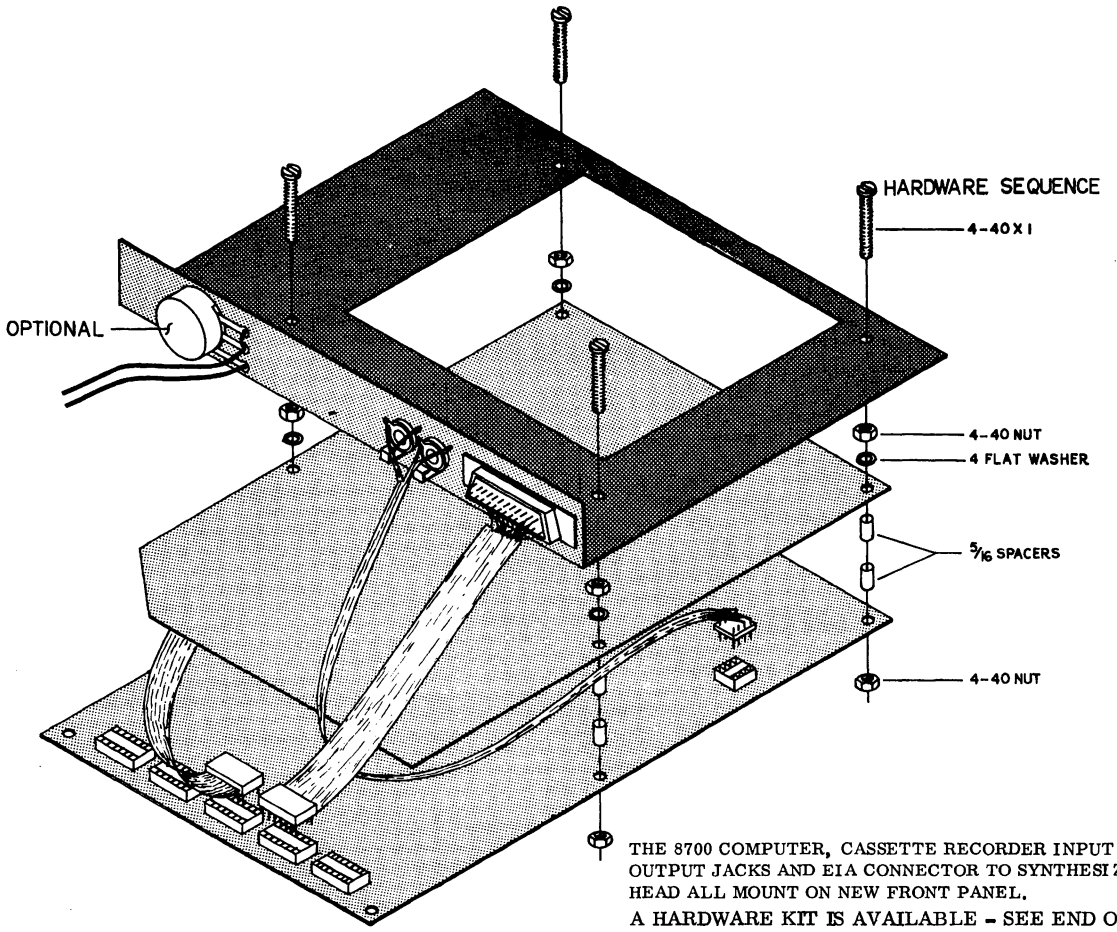


FIG.6



THE 8700 COMPUTER, CASSETTE RECORDER INPUT & OUTPUT JACKS AND EIA CONNECTOR TO SYNTHESIZER HEAD ALL MOUNT ON NEW FRONT PANEL. A HARDWARE KIT IS AVAILABLE - SEE END OF ARTICLE

FIG. 7

3) Clear the trigger flags (D6) of all notes in NTABLE (the output buffer).
4) Compare each entry in the input buffer (KTABLE) to each entry in the output buffer (NTABLE). If they are the same, set the trigger bit of the NTABLE entry and eliminate (zero) the entry from KTABLE. If all available outputs are used, or if all keys down find a home go to 1.
5) Place the remaining input buffer entries in output buffer locations which do not currently correspond to a down key (those in which D6 is cleared). When all input data has been placed or all channels available have been used go to 1.

There are a number of subtle implications here and unfortunately not enough space to cover them all.

A couple of really important ones are that if we think of "new" notes as ones corresponding to keys that were just pressed, this method tries to place those new notes in output channels which at some point in the past were already producing those notes.

This prevents a string of identical eighth notes (for example) from being assigned to different outputs each time they're used. Notes, once assigned, tend to stay assigned regardless of other keyboard activity - they don't move around in a totally unpredictable fashion as with some analog multi-note keyboards.

It also means that once the number of output channels available is "used up" by down keys that need to be placed, all other keys that are down are simply ignored (this is exactly what you want).

One important aspect of the above is that the program must "know" how many output channels are available to it, otherwise there is the possibility that notes may be assigned to non-existant channels (ones that have no corresponding hardware, not too bad in itself) and further (the really bad part) future activations of the note will be assigned <u>again</u> to these non-existant outputs - producing "dead" synthesizer keys that seem not to be doing anything.

Memory location $00EA contains the number of synthesizer channels available, more on this shortly.

THE PROGRAM

Shown on the next page is a disassembled listing of POLY 1.0.

Because, again, of space limitation we cannot re-print a fully documented version of POLY 1.0. It is supplied with the assembly and using manuals for the 8781 QuASH.

POLY 1.0 is also available in 8700 compatible cassette-tape form for $4.00.

LOADING AND INITIALIZING POLY 1.0

If you have a cassette interface on your 8700 and the POLY 1.0 tape, loading is simply a matter of connecting your tape recorder to the cassette input connectors on the 8700 and loading the tape using the following entry sequence:

0-0-0-0-0-0-F-F-0-0-1-1-TAPE

If you don't have the CS-87 option, you must enter the code manually from the 8700 keyboard.

POLY 1.0

By John S. Simonton, Jr.


```
06-  A9 00      LDA  #$00
08-  A2 18      LDX  #$18
0A-  95 CF      STA  $CF,X
0C-  CA         DEX
0D-  D0 FB      BNE  $080A
0F-  A2 08      LDX  #$08
11-  B5 D7      LDA  $D7,X
13-  18         CLC
14-  75 DF      ADC  $DF,X
16-  8D 00 09   STA  $0900
19-  9D F7 09   STA  $09F7,X
1C-  A0 04      LDY  #$04
1E-  88         DEY
1F-  D0 FD      BNE  $081E
21-  CA         DEX
22-  D0 ED      BNE  $0811
24-  A2 08      LDX  #$00
26-  A9 00      LDA  #$00
28-  95 CF      STA  $CF,X
2A-  CA         DEX
2B-  D0 FD      BNE  $0828
2D-  A2 08      LDX  #$00
2F-  2C 10 08   BIT  $0810
32-  30 FD      BMI  $082F
34-  2C 10 08   BIT  $0810
37-  30 0F      BMI  $0848
39-  50 F9      BVC  $0834
3B-  AD 10 08   LDA  $0810
3E-  95 CF      STA  $CF,X
40-  CD 10 08   CMP  $0810
43-  F0 FD      BEQ  $0840
45-  CA         DEX
46-  D0 EC      BNE  $0834
48-  E6 E8      INC  $E8
4A-  A5 E8      LDA  $E8
4C-  8D 20 08   STA  $0820
4F-  EA         NOP
50-  EA         NOP
51-  EA         NOP
52-  A5 EA      LDA  $EA
54-  85 EB      STA  $EB
56-  A2 08      LDX  #$08
58-  A9 BF      LDA  #$BF
5A-  35 D7      AND  $D7,X
5C-  95 D7      STA  $D7,X
5E-  CA         DEX
5F-  D0 F7      BNE  $0858
61-  A9 09      LDA  #$09
63-  85 E9      STA  $E9
65-  C6 E9      DEC  $E9
67-  F0 23      BEQ  $088C
69-  A6 E9      LDX  $E9
6B-  B4 CF      LDY  $CF,X
6D-  F0 1D      BEQ  $088C
6F-  A2 09      LDX  #$09
71-  CA         DEX
72-  F0 F1      BEQ  $0865
74-  98         TYA
75-  55 D7      EOR  $D7,X
77-  0A         ASL
78-  0A         ASL
79-  D0 F6      BNE  $0871
7B-  98         TYA
7C-  15 D7      ORA  $D7,X
7E-  95 D7      STA  $D7,X
80-  C6 EB      DEC  $EB
82-  F0 31      BEQ  $08B5
84-  A6 E9      LDX  $E9
86-  A9 00      LDA  #$00
88-  95 CF      STA  $CF,X
8A-  F0 D9      BEQ  $0865
8C-  A9 00      LDA  #$00
8E-  A2 09      LDX  #$09
90-  CA         DEX
91-  F0 22      BEQ  $08B5
93-  B4 CF      LDY  $CF,X
95-  F0 F9      BEQ  $0890
97-  95 CF      STA  $CF,X
99-  A2 09      LDX  #$09
9B-  CA         DEX
9C-  F0 17      BEQ  $08B5
9E-  A9 40      LDA  #$40
A0-  35 D7      AND  $D7,X
A2-  D0 F7      BNE  $0890
A4-  A9 80      LDA  #$80
A6-  35 D7      AND  $D7,X
A8-  95 D7      STA  $D7,X
AA-  98         TYA
AB-  15 D7      ORA  $D7,X
AD-  95 D7      STA  $D7,X
AF-  C6 EB      DEC  $EB
B1-  F0 02      BEQ  $08B5
B3-  D0 D7      BNE  $088C
B5-  20 00 FF   JSR  $FF00
B8-  C9 04      CMP  #$04
BA-  B0 03      BCS  $08BF
BC-  4C 06 00   JMP  $0006
BF-  C9 08      CMP  #$08
C1-  B0 05      BCS  $08C8
C3-  A9 2E      LDA  #$2E
C5-  4C 00 00   JMP  $0000
C8-  4C 0F 00   JMP  $000F
```

The cassette version of this program loads all of page zero of memory (its total requirement) and in the process initializes a couple of things that you will need to care for manually if the cassette is not available. When entering manually, be sure to set the number of outputs to correspond to the number you have available. For example, assuming that you have a system with a single QuASH, the number of channels available should be set to 4 using the following computer keyboard sequence:

RESET-0-0-E-A-DISP
0-4-ENTER

The tape version initializes the number of outputs at the most likely number of 4. If you want to use less channels (because of lack of modules, say) or have a system with more, do it as was shown above.

When entering the program manually, make sure the decimal mode flag in the status register is cleared by using this sequence:

RESET-0-0-F-F-DISP
0-0-ENTER

This is automatically taken care of when the tape version is loaded.

USING POLY 1.0

With everything connected, loaded and initialized, we're ready to begin making music. Go to the beginning of the program and begin running it.

RESET-0-0-0-6-RUN

If everything is working properly, we will see the 8700 displays counting quickly, incrementing by one for each scan of the keyboard. All of the QuASH outputs should be at a very low output voltage (the program initializes them as zero) and the trigger flags for each channel should be cleared.

As we press synthesizer keys, QuASH channels should "come alive" and produce control voltages corresponding to the keys that POLY 1.0 has assigned to them. The trigger flags should be set if the key corresponding to the channel is currently down and clear when the key is released.

TWO MORE FEATURES OF POLY 1.0

While POLY is running, touching any of the keys from 0-3 on the 8700 keyboard (the first row of keys) causes the system to clear all QuASH channels to zero and wait for new data to be assigned. You'll figure out what this is good for as you become familiar with the system.

Maybe more importantly, touching any of the keys 4-7 (the second row on the 8700) provides a tuning function and causes all QuASH channels to produce the same note with the trigger flags set, allowing all oscillators to be set to the same pitch. The note produced corresponds to the 2nd C on a standard configuration 3 octave keyboard. THE CHANNELS MUST BE CLEARED AFTER TUNING by touching the first row of 8700 keys.

THE SYNTHESIZER

There are an almost unlimited number of ways to use the multiple control voltage produced by the QuASH and POLY 1.0. You may want to use multiple VCO's mixed into a single voicing circuit, (See figure 9), or what amounts to a complete synthesizer for each control channel or anything in between, (See figure 10).

A word of advice: in your beginning stages of learning to use this system, you should try to stick to configuration in which all of the channels are producing the same "type" of sound - as close to identical as possible. As your skills progress and you develope a feel for how POLY 1.0 is going to massage data you can work up to using some output channels to set VCO pitches while

others control filter parameters (just an example - the number of possible combinations is extraordinarily large).

POLY 2.0 is under developement and features the use of some QuASH channels as software controlled envelope generators, reducing the need for lots of these hardware modules.

POLY 3.0 provides for computer storage of sequences of chordes or notes.

ONLY POLY 1.0 IS AVAILABLE NOW. The others are still a couple of months away. I mention them only because I want to make sure that we all understand that the nature of this new musical tool is a function of the program that is running and not so much of the hardware that it uses.

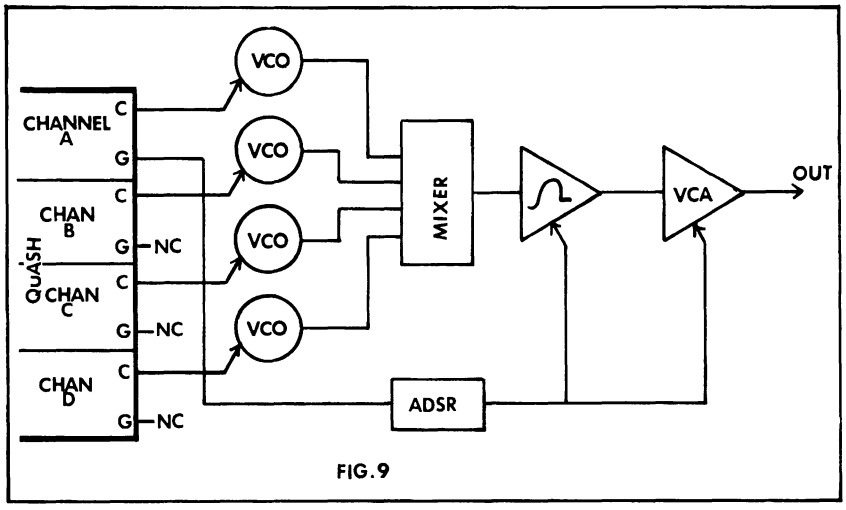


FIG. 9

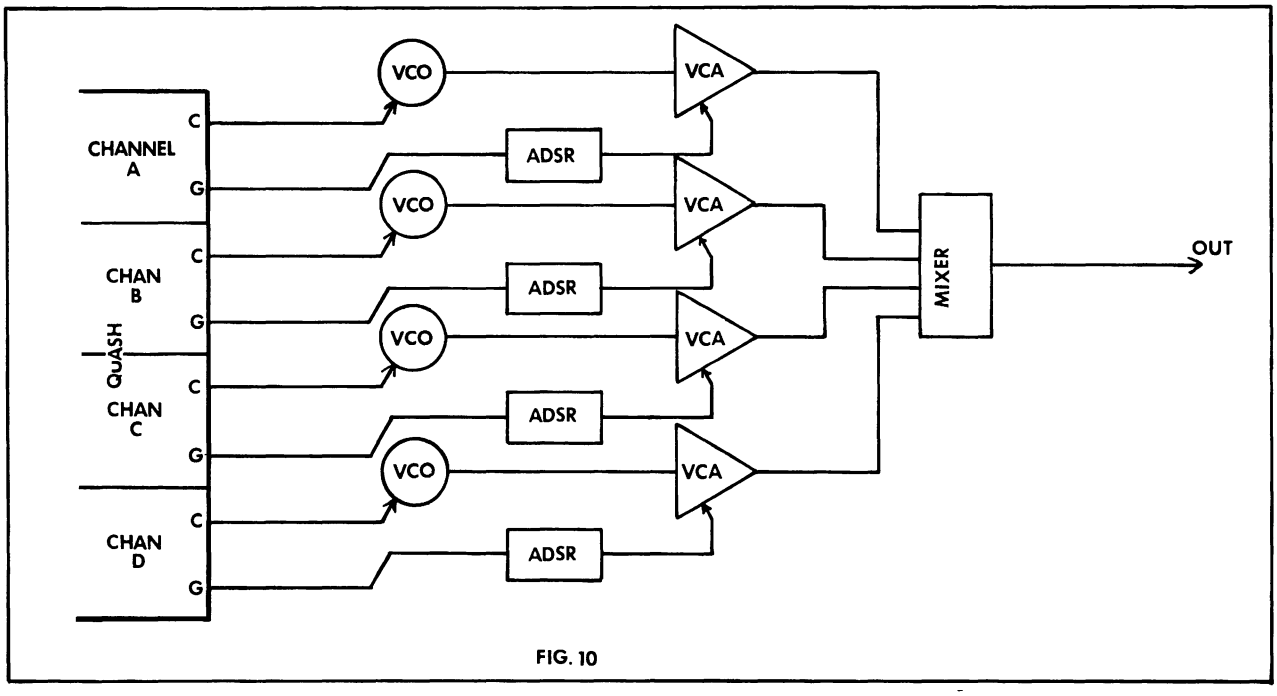


FIG. 10

LAB NOTES: MUS 1 with the new miracle ingredient – stg

by John S. Simonton, jr.

With the exception of the bare-bones listing of POLY 1.0 that ran in the last issue, we haven't looked at any software- mainly because there was little to examine.

But MUS1 was just recently finalized, so that situation is beginning to change.

MUS1, for the benefit of those of you who haven't been waiting for it for the last six months, is what many would call "system firmware"- and since that has the sort of technical ring to it that tends to make things interesting, we'll call it that, too.

In almost any computer application there are some programs which, for one reason or another, are best handled as firmware- a name that these days means not software (which must be loaded from some storage media external to the computer) and not hardware (a permanently wired collection of gates, etc. which cause a specific, set sequence of actions to take place) but something betwixt and between; most usually, software that is contained in a PROM somewhere.

The most obvious firmware is a monitor program such as PIEBUG. Since this program is the thing that allows for the entry of data and instructions into the memory of the computer in the first place (as well as usually providing whatever de-bugging and editing features the designer thought were important and/or had room for), it is at least inconvenient to have to load it every time it is needed. Much better to have it in a dedicated PROM where it is always available for immediate use.

The firmware of MUS1 is roughly analogous. These are universally useful routines that, with rare exceptions, will be used with everything we do musically. It's a waste of time and resources to have to load them to RAM from tape (or worse yet, manually) every time they're needed. A PROM is their happiest home. In our 8700 Computer/Controller, MUS 1 is a 1702A PROM that occupies the address range $D00-$DFF (IC-17).

Examples? OK, the keyboard reading routine (LOOK). It isn't particularly long or complicated (a little over 30 bytes) but we're going to need it every time we turn on the system- even if it isn't used to read the keyboard, it's the thing that our protocols dictate will be the tempo-determining element in the system (based on the clock rate of the encoder). At some future date the occasion may arise when we can examine this in detail. Today, it's not the point.

The QuAsh drivers (called NOTE)- same thing- we're going to need them for almost everything we do. Why bother to load them?

In addition to these two routines, MUS1 also contains:

INIT: an initialization routine that takes care of setting various variables and buffer areas to a known, acceptable state (as opposed to the random numbers they will contain when power is first applied.)

POLY: essentially the polyphonic (I still prefer polytonic) allocation algorhythm from POLY1.0, except refined somewhat to take less memory space.

TRGN: The new miracle ingredient- Software Transient Generators (STG). A routine that will serve as a software substitute for ADSRs.

OPTN: A very simple option selecting program that allows the remaining firmware of MUS1 to be tied together into a 16 voice polyhonic synthesizer with or without software transient generators- without having to lead any additional software (though several parameters will need to be initiallized manually).

All of this is pretty straight-ahead code that should be understandable from the documented listing that appears at the end of this article- you may need to refer back to previous articles in this series for background information; "In Pursuit of the Wild QuAsh" (reference Polyphony, July '77) and "What the Computer Does" (reference Polyphony 4/76) would be particularly useful ones.

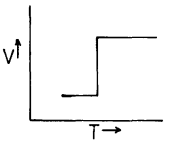
Two exceptions, NOTE and TRGN, need some additional explanation - they introduce some new ideas.

In an embryonic form, NOTE was a part of POLY1.0. It is the responsibility of this routine to take individual entries from the output buffer area (NTBL), add to it the corresponding entry from the Transpose buffer area (TTBL)and output the results to the QuAsh channels. Some aspects of the significance of the addition that takes place will be seen when we look at TRGN- for now, it will suffice to say that this will be an extraordinarily handy convention in a number of cases.

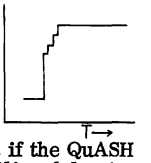
A more important function of NOTE is to make sure that what comes out of the QuAsh channels has no annoying glitches that may be artifacts of the D/A and multiplexing process. In an earlier story, we looked at one of the annoyances - the fact that our 8780 D/A, though quick, takes a finite amount of time to change from one value to the next and if appropriate settling time is not allowed between writes to the QuAsh channels we will be able to hear the changes as a slight "buzz" in each of the channels. The solution here is to output the data first to a "dummy channel" that is occupied solely by the D/A, with no corresponding QuAsh, followed immediately by a write of the identical information to an output which does correspond to a QuAsh channel. The first write allows the D/A to settle while the second strobes the settled output into the appropiate QuAsh channel.

And here we come face to face with the next problem; the QuAsh really need some settling time since they are at their heart nothing more than an RC circuit.

As long as we are thinking in terms of small systems(8 output channels or less) this is not a big problem since it can be dealt with simply by delaying after writing to one QuAsh but before setting up the next. If the delay is not long enough, we will hear changes from one value to the next not as an instantaneous change, but rather as a series of steps from the initial value to the final one:



We want this figure a



But if the QuASH settling delay is not long enough, will get this.

In larger systems, this constant delay approach is not a practical solution because there is not enough time during alternate "dummy" scans of the keyboard (the time which our conventions allow for processing, output driving, allocations, etc.) to allow all of the output channels the luxury of a delay. The time comes for the keyboard to be read again (or other things to happen) and the processor is still busy waiting for all of those QuAsh to get to the right value.

The key to the solution of this problem is to notice that there is really only one set of circumstances under which the long QuAsh-settling delay is required, and that's when the output of one of these channels must change from one value to another (which happens only a small percentage of the time) and then, only when the glide of the channel is turned off. (if

the glide is on, its integrating action will smooth out the steps; and, in fact, a short write time is preferable here since it will serve to increase the time required for the glide.)

The actual solution is what I feel we should call "DYNAMIC QuAsh DRIVERS" -a small block of programming, more or less in the middle of NOTE.

This part of the program first checks to see if the glide control bit (the most significant bit of the data just written to D/A and S/Hs) was turned on or not. If we are in "glide mode," no delay is required so the program immediately goes to see if there are any channels left to write; if there are, it services them.

If the glide is not on, we have a candidate for dynamic operation so the dynamic mode switch is checked (more later) and if this option is selected the current data is compared to the data that was previously written to this channel (requires a new table that we've generated called "LAST") and if they're different (a change), the program goes into the delay that allows the output of the QuAsh to instantaneously (apparently) step from its previous value to the next one. The new value is saved in LAST (for use next time) and if there are more channels to do- it does them.

SOFTWARE TRANSIENT GENERATORS

Here we begin, for the first time, to replace some of the elements that constitute traditional synthesizer hardware with software that performs the same function (hopefully as well, or better) with less costly hardware. STGs are a good place to start because they're not super difficult to implement.

Just like their hardware equivalents, STGs respond to a note which has just been triggered (pressed on the keyboard) by producing a voltage that rises at a controlled Attack rate. After reaching some peak value, the voltage then drops at a Decay rate until it reaches a pre-set Sustain level where it stays as long as the note remains triggered. When the key is released, the voltage drops to its lowest level at the Release rate.

Computing the number which represents the current value of the transient is only slightly more complicated than adding, subtracting and comparing.

Unlike an ADSR, an STG has no knobs to set, in their place you enter numbers setting Attack rate, etc. into the computer.

Perhaps the biggest problem having to do with STGs is deciding where they should come out. Oh, the QuAsh channels, obviously; but which ones? Of the numerous possibilities, we've selected the convention of having pitch setting voltages (those that correspond to notes) and transient voltages come from alternate QuAsh channels, primarily because this will work nicely with some stuff under development (or consideration, at least), without making obsolete all of the hardware that we've accumulated up to now.

This implies two distinct modes of operation; the first in which the STGs are not asserted and POLY assigns notes to sequential QuAsh channels; and, the second mode (STGs on), in which notes are assigned by POLY* to the odd number QuAsh channels (first, third, etc.) while transients are produced at the even number outputs (second, fourth, etc.).

The note produced at the first QuAsh output has a corresponding transient happening at the second output, and so on. Just as if the trigger from the first channel were patched to the input of an ADSR whose output was somehow tied to the output of the second QuAsh channel.

This would seem a good place to mention (in case it's not already obvious) that in this implementation all of the STGs produce the same kind of transient, and for the kinds of things that we're doing now, this is how it should be. It may also be worth mentioning that while the transients are all the same, they are totally independent where following the triggered and released states of their respective note channels is concerned.

There are also some internal details which muddy the STG waters. For instance, a key that is currently down may require a transient function that is either in the Attack cycle (increasing) or Decay/Sustain cycle (decreasing or holding) depending on its past history (had it already peaked?). Somewhere we need to save information on which cycle the transient is actually in.

Another, somewhat interrelated, problem concerns the smoothing of the transient waveform. Under most conditions, the glide of the QuAsh channels that are being used as transient outputs should be turned on so that a smoothly increasing or decreasing function is produced. But, the glide can't always be on because that would limit the maximum attack rate.

Without having the space to cover it entirely, I can only state that the solution to both of these difficulties lies in the use of the Transpose table and remembering that the data stored in TTBL entries is added to the output parameter in NTBL (where we're storing the actual current value of the transient) before the output operation takes place. Note also that while the data in NTBL is manipulated extensively by POLY and TRGN (as they calculate, allocate, - regurgitate?) TTBL is untouched by computer hands, and this makes it an ideal place to save control type functions. Not only transpositions, but a place that glide and trigger bits and such can be permanently set.

These locations are so handy for this application that in TRGN they have been re-named CWRD (Control-Words...but do not be confused, this is still our old friend TTBL and has no relationship at all to the System Control Word-CTRL) and it is here that we keep track of the A/D/S state of each of the transient channels.

Also, to help me keep things straight in my own mind, the NTBL bytes that are used to store the current value of the transient have been re-named PARM (parameter); but, again, this is the same physical area as NTBL.

* Note that POLY checks to see if the STGs are turned on as it assigns notes to outputs.

NOW, HOW DO WE USE ALL THIS?

Perhaps the best way to begin an essay on how to use MUS1 is to state one of the functions that it was devised to perform

As you are no doubt beginning to realize, we've carefully developed a system that will have applications far beyond what we've discussed to this point. It's complex; and while the complexity implies unmatched versatility, it undeniably has its intimidating aspects.

At one level of use, MUS1 should reduce this intimidation by giving the user an instrument with a specific (though within certain limits alterable) personality the instant that it's turned on, without having to hassle around with loading any additional programs (success) or variables (well...)

Also, these program modules should be written so that they easily interface with future expansions of the system, either hardware or software, so that, when needed, they can be accessed by programs offering distinctly different personalities (success here maybe- only time will really tell).

While we've reduced the intimidation, we've not eliminated it entirely because even when using MUS1 as a stand-alone personality there are some variables which must be initialized before you begin to play- some information that the system must have in order to operate properly. This data could be part of the PROM, but not without significantly compromising versatility.

For instance, we've mentioned in passing a couple of times the System Control Word-CTRL. This is a single word in the com-

puter's RAM memory at location $0E8.

It is most helpful to visualize CTRL as a collection of eight "switches", each bit representing one switch. To MUS1, only two of these switches have any significance- D7, which turns the STGs on and off, and D6, which enables or disables the dynamic mode option. The rest are reserved.

Every time you power up the system, CTRL must be set so that the desired options are selected- there is no default setting that is part of MUS1. If you want dynamic mode (which you should, for now) then bit 7 should be turned on. If you want STGs, bit 8 must be set.

The 4 possible combinations of these 2 bits then have the following significance:

binary	hex	action
00000000	$00	STGs off; dynamic mode off
01000000	$40	STGs off; dynamic mode on
10000000	$80	STGs on; dynamic mode off
11000000	$C0	STGs on; dynamic mode on

CTRL is not the only variable which must be initialized manually. There's also:

```
EXTERNALLY INITIALIZED VARIABLES
================================
 LOC. LABEL          USE
--------------------------------
 0E8  CTRL  SYSTEM CONTROL WORD
            D7 SET TURNS ON
            TRANSIENT GENERATORS
            D6 SETS DYNAMIC MODE

 0E9  ODLY  SETS OUTPUT DELAY;
            IN DYNAMIC MODE $20
            RECOMMENDED

 0EA  OUTS  NUMBER OF HARDWARE
            SUPPORTED CONTROL
            CHANNELS AVAILABLE

 -- AND TRANSIENT PARAMETERS --

 0BA  ATCK  ATTACK RATE
 0BB  DCY   DECAY RATE
 0BC  SUST  SUSTAIN LEVEL
 0BD  RELS  RELEASE RATE
 0BE  PEAK  PEAK VALUE -SEE TEXT

            RATES: $01 (SLOW)
                   $3F (FAST)
            LEVEL: $01 (MINIMUM)
                   $3F (MAXIMUM)
--------------------------------
```

Most of these are easily understood or have been examined in the past, so we won't go into any great detail. A few points are worth mentioning, however.

ODLY- this is a number that represents the delay that the QuAsh drivers will use, when required. For normal use, a value in the range of $20-$30 is most appropriate.

OUTS- this variable tells the POLY subroutine how many output channels it has to work with, so that notes don't get lost; we talked about this last time. Now we need to notice that when the STGs are asserted we should think of the QuAsh channel that is producing the transient as simply an extension of the channel producing the note. In other words, the two QuAsh channels constitute a single "hardware supported" channel. A single QuAsh represents two such channels.

ATCK/DCY/SUST/RELS- When the transient generators are turned on, we also need to enter the attack, delay, sustain and release parameters that we want produced. These four entries should need little explanation other than the examples which follow shortly; their range is from $01-$3F, with $01 representing the lowest rate or level and $3F the highest.

PEAK- this fifth transient parameter needs a little extra attention. PEAK has only one use; it determines whether the transient produced is going to be percussive (quickest possible attack and full ADSR segments) or non-percussive. In the non-percussive mode, the glide is on for all segments of the transient and the Decay and Sustain states of the transient are eliminated entirely.

In fact, there is only one bit in the word PEAK that is changed to select one of these two options- the most significant bit. The remaining seven bits should (for now- until you have a real feel for what's happening) be set to $3F (00111111 in binary). If the most significant bit of this word is cleared, you're in percussive mode. If the bit is set (so that PEAK contains $BF - 10111111 in binary) you are in non-percussive mode.

The differences between the two are great. Assume for a moment that we have set the ADSR parameters at $3F/$04/$20 /$01 respectively (fastest attack/moderated decay/medium sustain/slowest release) and that we are only going to change the PEAK parameter. If PEAK contains $3F(percussive mode), a 'scope display of the transient will look something like this:

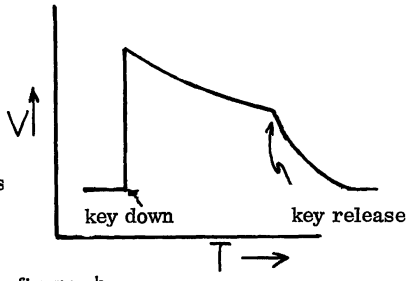


figure b

Setting PEAK to $BF (non-percussive) produces this result:

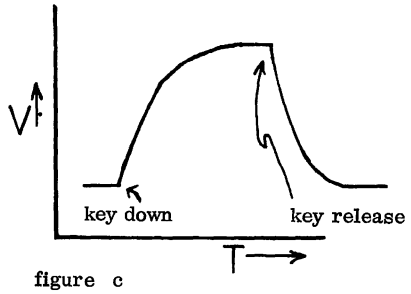


figure c

Because the glide is now on during the entire attack cycle and the Decay and Sustain portion of the transients are eliminated. Straightforward stuff, really.

We need to cover an example of system set-up before we wind up, but first must notice that the effect of having the PEAK parameter are far more far-reaching than we've been able to cover in detail. A quick example:

ADSR parameters set to $10/$04/$20 /$01 and PEAK containing $3F will produce this kind of transient:

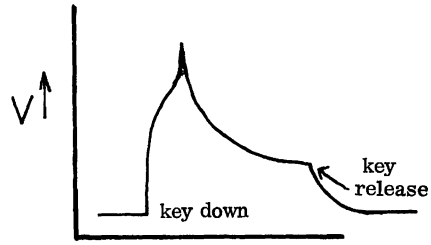


figure d

which, when heard, starts out with a non-percussive kind of "swell" with a percussive "pip" added at the last instant before the transition to the Decay and Sustain cycles. This would seem to be a unique and useful transient that isn't produced by traditional ADSRs.

Along the same lines, the TSGs can be considered to be "better" than our hardware ADSRs in that they need not finish the Attack cycle before transitioning to the Release state. If a key is released before its transient has gone all the way to PEAK, the transient immediately switches to the release state. This is frequently called "muting' and it offers the possibility of effective control of espression directly from the AGO keyboard.

A SUMMARY, OF SORTS

So, we've gotten our hands on a MUS1 PROM and are ready to start doing things. What has to be done first? Really very little.

First, the System Control Word, Output Delay and number of hardware channels available must be set. For example:

keystrokes	explanation
0-E-8-DISP	sets monitor pointer to $E8-CTRL
C-0-ENT	sets $E8-asserts STGs dynamic mode
3-0-ENT	sets ODLY value
0-2-ENT	sets output channels at 2

these entries define the personality of the instrument as a 2 voice polyphonic synthesizer (notes from channels A & C) with software transient generators (which appear at QuAsh channels B & D) .

Next, we must set the transient parameters to the desired values:

keystrokes	explanation
0-B-A-DISP	sets monitor pointer to $BA- ATCK
3-F-ENT	sets shortest attack
0-4-ENT	sets moderate decay
2-0-ENT	sets moderate sustain
0-1-ENT	sets slowest release
3-F-ENT	percussive mode

and you may recognize these parameters as being those that we examined in the illustration earlier.

Finally, we simply begin running the program:

keystrokes	explanation
D-0-0-DISP	sets monitor pointer to beginning of OPTN
RUN	presto- the program runs

A typical patching configuration that would be consistent with these entries would look something like this:

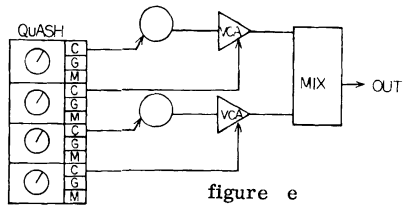


figure e

Oh, yes- I almost forgot. OPTN, like POLY 1.0, uses the 8700 keyboard to control two important functions. While OPTN is running, touching key 0 of the control keyboard will cause the entire system to be re-initialized. Not the entries that we made manually- those remain unchanged, but all the notes and transients go immediately to zero level.

Similarly, touching key #1 produces a tuning function that makes the synthesizer respond as if all the channels were seeing the second C on a three octave keyboard held down. The transients go, the notes play, etc. After tuning, be sure to re-initialize the system by touching control key #0.

I prefaced one of the earlier paragraphs with "at one level of use." In all of the preceding words, that's all that we've examined- one level of use (the simplest and most obvious level, at that.). I've also referred in the past to "software modules" which can be strung together in different ways (just as can hardware modules) to produce different effects and personalities. MUS1 is the first set of these modules.

With more regret than you can imagine, I haven't the space here to go into all of the implications of this (even if I knew them all, which I'm sure I don't).

Providing you're more than just casually interested, you should spend some time trying to understand how MUS1 works internally (there are numerous different entry points to the routines that we haven't covered - for instance). I believe that the time investment will be wisely made.

```
                        ;**********************************
                        ;*                                *
                        ;*             MUS1               *
                        ;*                                *
                        ;*       BY JOHN SIMONTON         *
                        ;*(C) 1978 PHIA ELECTRONICS, INC*
                        ;*      ALL RIGHTS RESERVED       *
                        ;*                                *
                        ;*                                *
                        ;*   SYNTHESIZER SUBROUTINES      *
                        ;*         ++ AND ++              *
                        ;*  MULTIPLE OPTION POLYPHONIC    *
                        ;*   ALLOCATION PROGRAM WITH      *
                        ;*     SOFTWARE TRANSIENT         *
                        ;*         GENERATION             *
                        ;*                                *
                        ;**********************************
                        ;
                        ;          OPTN
                        ;POLYPHONIC SYNTHESIZER / OPTION
                        ;       SELECTION
                        ;**********************************
                        ;
                        ;
                        INIT .DL 0D21
                        POLY .DL 0D71
                        TRGN .DL 0DC3
                        NOTE .DL 0D2B
                        DECD .DL 0F00
                        FILL .DL 0D52
                        DISP .DL 0820
                        CLCK .DL 00BF
                        ;
                        ;OPTION TIES MUS1 FIRMWARE
                        ;TOGETHER INTO A POLYPHONIC SYNTH
                        ;WITH OR WITHOUT TRANSIENT GENER-
                        ;TION; W/WO DYNAMIC QUASH DRIVERS
                        ;
                        ;ALSO USES PIEBUG DECODE AND
                        ;ASSIGNS KEY #0 AS SYSTEM CLEAR
                        ;AND #1 AS TUNE - EQUIVALENT TO
                        ;ALL CHANNELS 2ND "C" ON KBD DOWN
                        ;
0D00-   20 21 0D  OPTN  JSR INIT     ;ZERO ALL BUFFS
0D03-   20 71 0D  LOOP  JSR POLY     ;ALLOCATE CHANS
0D06-   20 C3 0D        JSR TRGN     ;NEW TRANSIENTS
0D09-   20 2B 0D        JSR NOTE     ;OUTPUT-READ AGO
0D0C-   A5 BF           LDA *CLCK    ;GET CLOCK VALUE
0D0E-   8D 20 08        STA DISP     ;RAZZ-MA-TAZZ
0D11-   20 00 0F        JSR DECD     ;CHECK COMMANDS
0D14-   C9 01           CMP 01       ;0?  1?  >1?
0D16-   30 E8           BMI OPTN     ;0; CLEAR ALL
0D18-   D0 E9           BNE LOOP     ;>1; KEEP ON
0D1A-   A0 5C           LDY 5C       ;1; TUNE 2ND C
0D1C-   20 52 0D        JSR FILL     ;KEYS ALL DOWN
0D1F-   F0 E2           BEQ LOOP     ;BRANCH ALWAYS
                        ;
                        ;          INIT
                        ;  INITIALIZATION ROUTINE
                        ;**********************************
                        ;
                        CTRL .DL 00E8
                        TBEG .DL 00BF
                        ;
                        ;INIT CLEARS INPUT BUFFER (KTBL)
                        ;OUTPUT BUFFER (NTBL) AND TRANS-
                        ;POSE BUFFER/CONTROL WORDS (TTBL)
                        ;HEXADECIMAL MODE IS SELECTED
                        ;
                        ;ENTER AT INT0 TO FILL TABLES
                        ;WITH CHARACTER FROM ACCUMULATOR
                        ;
0D21-   A9 00     INIT  LDA 00         ;PREPARE TO ZERO
0D23-   A2 28     INT0  LDX 28         ;SET POINT/COUNT
0D25-   D8              CLD          ;SET HEX MODE
0D26-   95 BF     INT1  STA *TBEG,X    ;ZERO BUFFER
0D28-   CA              DEX          ;POINT TO NEXT
0D29-   D0 FB           BNE INT1     ;SOME LEFT -LOOP
                        ;
                        ;       NOTEOUT/LOOK
                        ;16 CHANNEL QUASH DRIVERS AND AGO
                        ;   KEYBOARD READING ROUTINE
                        ;
                        ;**********************************
                        ;
                        CTRL .DL 00E8
                        ODLY .DL 00E9
                        KTBL .DL 00DF
                        NTBL .DL 00CF
                        TTBL .DL 00BF   ;ALSO CLCK
                        LAST .DL 00A9
                        S/H  .DL 09EF
                        D/A  .DL 0900
                        KBD  .DL 0810
                        ;
                        ;       *** NOTEOUT ***
                        ;    DYNAMIC QUASH DRIVERS
                        ;GETS NOTES TO BE PLAYED FROM THE
                        ;OUTPUT BUFFER (NTBL) AND ADDS
                        ;TRANSPOSING VALUE FROM TRANSPOSE
                        ;BUFFER (TTBL).  OUTPUTS RESULT
                        ;
0D2B-   A2 10     NOTE  LDX 10         ;SET POINTER
0D2D-   B5 CF     NO0   LDA *NTBL,X    ;GET NOTE
0D2F-   18              CLC          ;PREPARE AND
0D30-   75 BF           ADC *TTBL,X  ;ADD TRANSPOSE
0D32-   8D 00 09        STA D/A      ;LET D/A SETTLE
0D35-   9D EF 09        STA S/H ,X   ;WRITE TO S/H
                        ;
                        ;NOW THE DYNAMIC PART; IF GLIDE
                        ;IS ON, DELAY IS SKIPPED.  IF NOTE
                        ;IS SAME AS LAST PLAYED (IGNORING
                        ;CONTROL BITS D6 & D7) DELAY IS
                        ;SKIPPED.   IF NOT IN DYNAMIC MODE
                        ;AND NO GLIDE, DELAY ALWAYS TAKEN
                        ;
0D38-   30 0F           BMI NO2      ;GLIDE? NO DELAY
0D3A-   09 80           ORA 80       ;IGNORE FLAGS
0D3C-   24 E8           BIT *CTRL    ;DYNAMIC MODE ?
0D3E-   50 04           BVC DLAY     ;NO, JMP TO DELAY
0D40-   D5 A9           CMP *LAST,X  ;COMPARE TO LAST
0D42-   F0 05           BEQ NO2      ;SAME:SKIP DELAY
0D44-   A4 E9     DLAY  LDY *ODLY    ;GET DELAY VALUE
0D46-   88        NO1   DEY          ;DECREMENT DELAY
0D47-   D0 FD           BNE NO1      ;LOOP TIL DONE
0D49-   95 A9     NO2   STA *LAST,X  ;FOR NEXT TIME
```

```
0D4B-   CA            DEX          :POINT TO NEXT
0D4C-   D0 DF         BNE NO0      :SOME LEFT -LOOP
                      :
                      :LOOK WAITS FOR THE BEGINNING OF
                      :AN "ACTIVE" SCAN-BEGINS PUTTING
                      :THE NUMBERS OF KEYS DOWN IN SE-
                      :QUENTIAL IN-BUFF WORDS.  WHEN
                      :SCAN DONE REMAINING IN-BUFF IS
                      :ZERO'D.
                      :
0D4E-   E6 BF     LOOK INC *TTBL   :INCREMENT CLOCK
0D50-   A0 00         LDY 00       :PREPARE FOR CLR
0D52-   A2 08     FILL LDX 08      :SET UP POINTER
0D54-   AD 10 08  LK2  LDA KBD     :WAIT FOR
0D57-   30 FB         BMI LK2      :"ACTIVE" SCAN
0D59-   AD 10 08  LK3  LDA KBD     :GET KEY
0D5C-   30 0F         BMI DONE     :END SCAN? -CLR
0D5E-   2A            ROL          :STROBE TO D7
0D5F-   10 F8         BPL LK3      :D7=0,NO STROBE
0D61-   6A            ROR          :RESTORE DATA
0D62-   95 DF         STA *KTBL,X  :TO IN BUFFER
0D64-   CD 10 08  LK4  CMP KBD     :NOW WAIT FOR
0D67-   F0 FB         BEQ LK4      :NEXT KEY
0D69-   CA        LK0  DEX          :PNT TO NEXT BUF
0D6A-   D0 ED         BNE LK3      :SOME LEFT -LOOP
0D6C-   60        RTN  RTS          :LEAVE
0D6D-   94 DF     DONE STY *KTBL,X :ZERO IN-BUFFER
0D6F-   30 F8         BMI LK0      :BRANCH ALWAYS
                      :
                      :          POLY
                      : A LIMITED RESOURCE ALLOCATION
                      :         ALGORHYTHM
                      :**********************************
                      :
                      OUTT .DL 00EB
                      OUTS .DL 00EA
                      CTRL .DL 00E8
                      KTBL .DL 00DF
                      NTBL .DL 00CF
                      :
                      :POLY-FIRST HALF OF ALGORHYTHM
                      :IN THIS BLOCK DE-ACTIVATED CHANS
                      :ARE REACTIVATED IF THE DATA THEY
                      :CONTAIN APPEARS IN THE IN BUFFER
                      :
                      :D7 IN CTRL SET - ALTERNATE MODE
                      :D7 "   "   CLR - SEQUENTIAL MODE
                      :
0D71-   A5 EA     POLY LDA *OUTS   :# OF OUT CHANS
0D73-   85 EB         STA *OUTT    :USE AS COUNTER
0D75-   A2 10         LDX 10       :PREPARE PNT/CNT
0D77-   B5 CF     POL0 LDA *NTBL,X :GET NOTE
0D79-   F0 27         BEQ NWKY     :0-OLD KEYS DONE
0D7B-   29 7F         AND 7F       :CLEAR D7
0D7D-   09 40         ORA 40       :SET D6
0D7F-   A0 09         LDY 09       :PREPARE PNT/CNT
0D81-   88        LP0  DEY          :POINT NEXT KEY
0D82-   F0 12         BEQ NEXT     :DONE -NEXT NOTE
0D84-   D9 DF 00      CMP KTBL,Y   :SAME AS KEY?
0D87-   D0 F8         BNE LP0      :NO -NEXT KEY
0D89-   95 CF         STA *NTBL,X  :SAVE NOTE D6=1
0D8B-   C6 EB         DEC *OUTT    :ONE LESS OUTPUT
0D8D-   F0 33         BEQ OUT      :NONE LEFT-LEAVE
0D8F-   A9 00         LDA 00       :OR PREPARE AND
0D91-   99 DF 00      STA KTBL,Y   :ELIMINATE KEY
0D94-   F0 04         BEQ LP1      :& BRANCH ALWAYS
0D96-   29 BF     NEXT AND 0BF     :CLEAR TRIG (D6)
0D98-   95 CF         STA *NTBL,X  :& RESTORE NOTE
0D9A-   24 E8     LP1  BIT *CTRL   :ALTERNTE MODE?
0D9C-   10 01         BPL SKP1     :NO -DEC. ONCE
0D9E-   CA            DEX          :YES-DEC. TWICE
0D9F-   CA        SKP1 DEX         :POINT NEXT NOTE
0DA0-   D0 D5         BNE POL0     :SOME LEFT -LOOP
                      :
                      :NEWKEY - SECOND HALF.  KEYS DOWN
                      :ARE ASSIGNED TO OUTPUT BUFFER
                      :LOCATIONS WHICH ARE STILL DE-
                      :ACTIVATED
                      :
0DA2-   A2 10     NWKY LDX 10      :NTABLE PNT/CNT
0DA4-   A0 09         LDY 09       :KTABLE PNT/CNT
0DA6-   A9 40     NK1  LDA 40      :PREPARE MASK
0DA8-   35 CF         AND *NTBL,X  :NOTE TRIGGERED?
0DAA-   D0 0E         BNE NK3      :YES -GO TO NEXT
0DAC-   88        NK2  DEY          :POINT NEXT KEY
0DAD-   F0 13         BEQ OUT      :NONE LEFT-LEAVE
0DAF-   B9 DF 00      LDA KTBL,Y   :KEY NEEDS HOME?
0DB2-   F0 F8         BEQ NK2      :NO -GET NEXT
0DB4-   95 CF         STA *NTBL,X  :YES-PUT IN NOTE
0DB6-   C6 EB         DEC *OUTT    :ONE LESS OUTPUT
0DB8-   F0 08         BEQ OUT      :NONE LEFT-LEAVE
0DBA-   24 E8     NK3  BIT *CTRL   :ALTERNATE MODE?
0DBC-   10 01         BPL SKP2     :NO -DEC ONCE
0DBE-   CA            DEX          :YES-DEC TWICE
0DBF-   CA        SKP2 DEX         :POINT NEXT NOTE
0DC0-   D0 E4         BNE NK1      :SOME LEFT -LOOP
0DC2-   60        OUT  RTS         :RETURN
                      :
                      :          TRGN
                      :TRANSIENT GENERATOR PROGRAM
                      :
                      :*****************************
                      :
                      CTRL .DL 00E8
                      ATCK .DL 00BA
                      DCY  .DL 00BB
                      SUST .DL 00BC
                      RLS  .DL 00BD
                      PEAK .DL 00BE
                      NTBL .DL 00CF
                      PARM .DL 00CE
                      TTBL .DL 00BF
                      CWRD .DL 00BE
                      :
                      :       NTBL 00D0-00DF
                      :       TTBL 00C0-00CF
                      :
0DC3-   A5 E8     TRGN LDA *CTRL   :DO TRANSIENTS?
0DC5-   10 38         BPL RTN1     :NO  -RETURN
0DC7-   A2 10         LDX 10       :NTABLE PNT/CNT
                      :
                      :   A/D/S/R DETERMINATION
                      :ROUTINE PREPARES Y TO USE AS
                      :CONTROL WORD, GETS NOTE AND
                      :SHIFTS TRIG. TO CARRY, GETS
                      :CURRENT STATE (CS) PARAMETER.
                      :IF NOTE TRIG. NOT SET STATE IS
                      :RELEASE.  IF CS PARA IS POSI-
                      :TIVE STATE IS DECAY/SUSTAIN
                      :OTHERWISE, STATE IS ATTACK
                      :
0DC9-   A0 40     ADSR LDY 40      :PREPARE CWRD
0DCB-   B5 CF         LDA *NTBL,X  :GET NOTE AND
0DCD-   2A            ROL          :ROTATE TRIGGER
0DCE-   2A            ROL          :TO CARRY BIT
0DCF-   B5 CE         LDA *PARM,X  :GET CS PARA.
0DD1-   90 19         BCC RELS     :NO TRIG? -RLS
0DD3-   10 0B         BPL DS       :CS>0? -DECAY/S
                      :
                      :    ATTACK ROUTINE
                      :ADDS ATTACK PARAMETER TO CS PARA
                      :AND IF GREATER THAN PEAK
                      :SUBSTITUTES $3F AND SETS CONTROL
                      :WORD TO $40 (D6 SET - NO GLIDE).
                      :NOTE THAT CS PARA WILL BE >0
                      :WHEN NEXT CHECKED.
                      :
0DD5-   18        ATTK CLC          :PREPARE
0DD6-   65 BA         ADC *ATCK    :ADD ATTACK PARA.
0DD8-   C9 BF         CMP 0BF      : >PEAK
0DDA-   90 1B         BCC NEXT     :NO -PLACE PARA.
0DDC-   A5 BE         LDA *PEAK    :YES-PEAK VALUE
0DDE-   D0 17         BNE NEXT     :BRANCH ALWAYS
                      :
                      :    DECAY AND SUSTAIN ROUTINE
                      :NOTE THAT CARRY IS SET.  DECAY
                      :PARAMETER IS SUBTRACTED FROM
                      :CURRENT STATE PARAMETER.  IF
                      :RESULT IS LESS THAN SUSTAIN
                      :PARAMETER THEN SUST. PARA.
                      :BECOMES CURRENT STATE PARA.
                      :D6 & D7 OF CONTROL WORD SET
                      :
0DE0-   A0 C0     DS   LDY 0C0      :PREPARE CWRD
0DE2-   E5 BB         SBC *DCY     :SUBTRACT DCY
0DE4-   C5 BC         CMP *SUST    :>SUSTAIN?
0DE6-   10 0F         BPL NEXT     :PLACE PARA
0DE8-   A5 BC         LDA *SUST    :CS PARA=SUST
0DEA-   10 0B         BPL NEXT     :PLACE PARA
                      :
                      :    RELEASE ROUTINE
                      :MAKES SURE THAT CURRENT STATE
                      :GLIDE BIT IS SET (NOTE-MAKES
                      :CS NEGATIVE).  SUBTRACTS RELEASE
                      :PARA. FROM CURRENT STATE.  IF
                      :RESULT >0, MAKES CS & CWRD =80
                      :
0DEC-   38        RELS SEC          :PREPARE
0DED-   09 80         ORA 80       :SET CS GLIDE
0DEF-   E5 BD         SBC *RLS     :SUBTRACT RLS
0DF1-   30 04         BMI NEXT     :CS<0 -PLACE CS
0DF3-   A0 00         LDY 00       :CS>0 -DONE MAKE
0DF5-   A9 80         LDA 80       :CS=80,CWRD=0
                      :
                      :    NEXT
                      :PLACES CS PARA AND CWRD IN
                      :PROPER CONTROL CHANNEL OUTPUTS
                      :DECREMENTS POINTER (TWICE) AND
                      :IF NOT YET DONE LOOPS FOR MORE
                      :
0DF7-   94 BE     NEXT STY *CWRD,X :PLACE CONTROL
0DF9-   95 CE         STA *PARM,X  :PLACE CS PARA
0DFB-   CA            DEX          :DECREMENT POINT
0DFC-   CA            DEX          :AND AGAIN
0DFD-   D0 CA         BNE ADSR     :SOME LEFT -LOOP
0DFF-   60        RTN1 RTS          :RETURN
```

AFTERTHOUGHTS .....

It has been pointed out that some perhaps pertinent details have been omitted from the preceeding explanation of MUS 1.

The most prominent example is "why would you ever want to not have dynamic mode". The most probable reason is for special effects.

In general, the difference between special effects and noise is imagination. Contemporary musical lore is full of instances where a special effect resulted from an unsuccessful attempt to do something entirely different. Phil Spector' s original "flanging" effect, so popular today, was supposed to be voice doubling, but didn' t work.

In this same manner, there will be those who will be able to use the "step glissando" that results from too short a QuASH settling delay as a valid musical device.

Also, the dynamic mode requires an additional 16 byte table area that might easily be put to better use in some programs.

This same philosophy of maximizing versatility is responsible for the QuASH settling delay being an externally initialized variable. For the purpose of effect, there may be times when you want a short delay.

In addition to this, we have seen systems which were marginal in their power supply complement which would have a discernible pitch "blip" when keys were pressed with long delays (in the $30 - $40 range) - caused by the relatively heavy charging current producing a momentary dip in supply voltage. In these systems, a short term solution has been to decrease the QuASH driver delay to something on the order of $10. The long term solution is more power.

SEVERAL POINTS RELATIVE TO THE OPERATION OF THE STGs SHOULD BE MENTIONED.

QuASH GLIDE CONTROLS. The setting of the QuASH glide pots have an effect on the transients produced. In most cases, these controls will need to be advanced only slightly from their fully counterclockwise "off" position.

The most noticeable effect of different settings of the glides will be observed when the STGs are set to the percussive mode by PEAK.

When the most significant bit of PEAK is cleared, it will effect only on the last increment of the attack cycle. For all increments other than the last, the glide will be set. A detailed example will best illustrate this.

Assume that we have set the STG parameters as follows:

ATCK - 08
DCY - 04
SUST - 20
RELS - 04
PEAK - 3F

or in our more or less standard notation, $08/$04/$20/$04/$3F.

If we were able to disable glide entirely, and then scope'd the transient we'd see this:

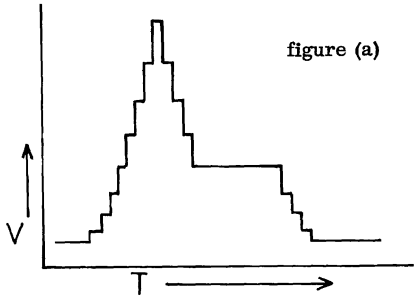


figure (a)

as the STG program counted up to the peak and then down to the sustain level before counting down to the base level when the key was released.

If we then enabled the glide and set them to a slightly advanced position and examined the same output we would find that this change had taken place.

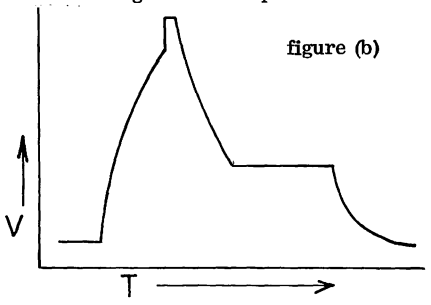


figure (b)

The integrating action of the glide circuitry has smoothed the steps of the Attack, Decay and Release, with the exception of the last Attack step where (as we have already stated) the glide is off under all percussive circumstances. In this specific case, the last glide-less increment will be hardly noticeable.

If, on the other hand, the glide is set to a long value (fully advanced, for instance) an examination of the waveform will show this:

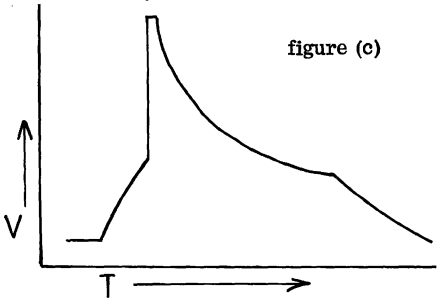


figure (c)

The heavy glide has slowed the waveform to the point that when the glide-less final increment comes it takes a much greater step (one that is completely noticeable, and unique). The heavy glide also has the effect of slowing the decay and release rates as shown.

It would also be appropriate to mention at this point that the instantaneous steps produced by the STG/QuASH combination is much faster than the maximum attack rate available from a 4740, or in fact from most ADSRs. Whereas a typical ADSR may have a minimum attack time of more or less 5 milliseconds, the QuASH in dynamic mode can step in a fraction of a millisecond.

This means that if there are any tendencies on the part of the VCA being used to have interaction between control and signal channels it will be aggravated when using STGs. We may hear "pops" and "thumps" that were not objectionable before. Probably the best solution here is to limit the response of the control voltage inputs of the VCA. In a 4710 Balanced Modulator, this means the addition of a small capacitor. Like this:

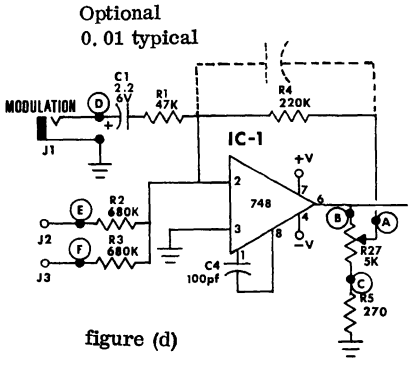


figure (d)

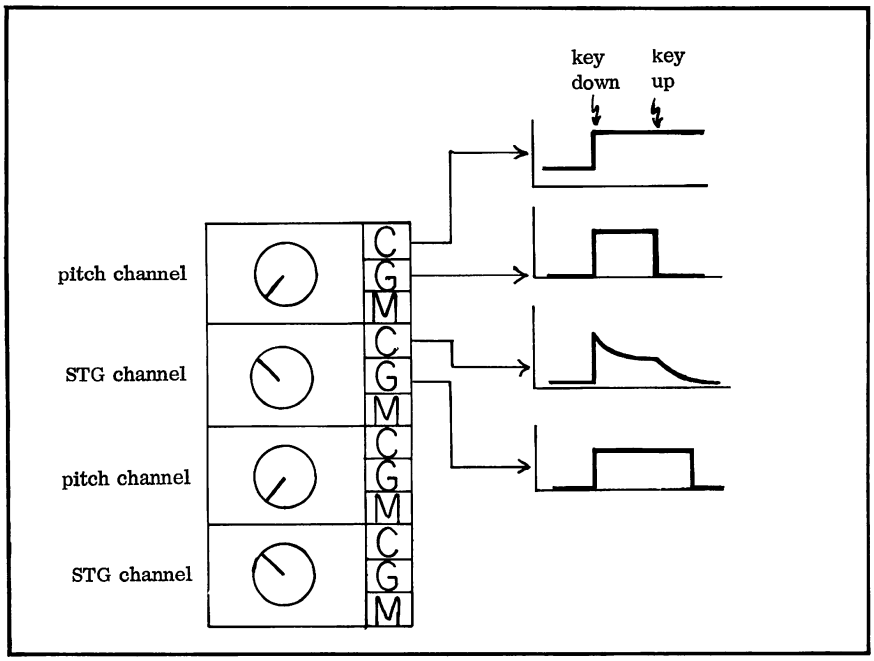
Another point to be considered is the fact that the output voltage from a QuASH channel doesn't go to zero - there may be some leakage from the VCA when it is supposed to be off. The easiest fix here is to re-adjust the Modulation rejection control of the VCA being used. In the 4710 this is R25. Be aware that this also limits somewhat the useable range of the D/A's TUNE control since wide variations in the setting of this control will affect the leakage from the VCA. Tuning changes on the order of 1/4 octave should not present any particular difficulties.

There is a point dealing with this which may not be immediately exploitable by many, but which should be mentioned in any case; the action of the trigger outputs of the QuASH channels which are being used as STG channels.

The trigger outputs QuASH channels which are being used to produce pitch setting voltages behave in the normal manner. When the AGO key

corresponding to that QuASH channel is being held down, the trigger is at a high state. When the key is released, the trigger goes low. A standard "gate" type response.

In a similar manner, the triggers of transient channels also go high as soon as the AGO key to which they correspond is pressed; but unlike the normal trigger, this level remains high until the software has completed the last increment of the Release cycle. In future hardware this will drive a "noise gate", a simple semi-conductor switch which completely quiets a channel that is inactive.



LAB NOTES! PINK TUNES

By: John S. Simonton, Jr.

As we begin this month's journey into the bizarre I should warn you that I'm operating in a somewhat altered state of consciousness.

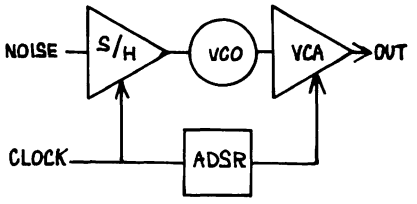
Oh, not from chemicals or nature's own, none of that. There's just some very nice color graphics going on the Apple II and the background music is a slightly oriental feeling 4 part harmony being composed by a P4700/J. It's really a most unique environment.

Wait. Composed by the synthesizer? Surely not, surely just something pre-recorded and played back.

Well, I suppose that I wouldn't attack someone who asserted that I composed the piece. I'd be flattered, but it wouldn't be entirely correct. I knew before the tune started what sort of texture (for lack of a better term) it would have. But I have no idea what exactly is coming next.

And that, in case you hadn't already guessed, is what we want to talk about this time. Computer programs that compose music.

Let's start at a very elementary level. Probably you've seen or connected synthesizer patches that look something like this:



It's a relatively common configuration in which, at regular intervals, the instantaneous value of the noise source is captured by the S/H and the resulting voltage used to set the pitch of the VCO. The ADSR and VCA give us some knobs to twiddle and control dynamics, but otherwise are just window dressing.

If you've done one of these, you know that the results are interesting, but certainly not a musical composition in the traditional sense. As a composing device, it's hard to know what the biggest fault is here, but certainly it must be the fact that there are no guarantees that the series of pitches produced are going to be equally tempered intervals (or any known tempering for that matter). In fact, you can almost guarantee that they won't be; the control voltages applied to the oscillator are completely random.

And therein lies the tale.

I don't believe that anyone is able at this point in human development to concisely explain what makes music "musical", but most folks that have thought about it seem to feel that "good" music (ugh, all the subjective terms) combines both order and disorder. Establish a pattern in the listener's mind ... then surprise him; pleasantly, preferably.

Like the "noise music" example above, any compositional program that we come up with today is in some way going to rely on a STOCHASTIC (big word for random) process. If it didn't, we wouldn't be writing the program that wrote the music; we'd be writing the music.

Our task then, is to bring order from disorder (in a very real sense, nothing less than reversing entropy) - but not completely. It isn't easy, but in an elementary form not as difficult as it may sound either because we now have at our disposal that wonder of wonders (which many right-thinking people say is Maxwell's Demon personified):

THE COMPUTER

By simply programming the computer to randomly select only pitches that are part of the equally tempered sequence, we've made a start, but in all honesty not much of one; still there is too much disorder. Low pitches followed as likely as not by very high ones, no identifiable key signature. It's still "noise music".

The quickest way to begin bringing the kind of order that we're looking for is to write a program that uses a random number not as the note, but as a "pointer" which is used to select one of a number of acceptable "candidate" notes from a previously entered table. We're using our intellect to select ahead of time only those notes which we know will harmonize with the rest of the notes which the computer is allowed to select. I've written a few of these kinds of programs. They're a little better than purely random notes, but not much. Still too much disorder.

There are a lot of tricks to bring rigorous order, like making random substitutions of candidate notes into previously entered melody lines. This kind of thing produces terriffic results, but it's not the computer doing most of the composition - you are.

Now comes the April issue of Scientific American and there, in Martin Gardner's consistently enlightening Mathematical Games column, is a piece on computer music. Well, not just computer music - as is usual, Mr. Gardner's mind ranges far and the column covers visual art and computer generated "landscapes" and fractal curves and the place of pink noise in "the meaning of it all". Very heavy. And buried in amongst it all is an algorithm conceived by Richard Voss (of IBM) for turning "white" random numbers "pink".

Don't let this "white" and "pink" business throw you. You're used to white noise and the pink noise that results when you filter it. We can think of the Voss algorithm as a filter for random numbers.

The realization of the Voss algorithm which is used in PINK TUNES (the program listing at the end of this column) can be likened to rolling a set of 5, four sided dice whose faces bear the numbers 0 - 3. We get the random number that we'll use as the pointer to the list of candidate notes by adding together the numbers on the exposed face of each die (I know, a 4 sided die won't have an upper face, that's not the point).

If we consistently rolled all 5 dice, we would still produce too random a number; even through, as any crap-shooter can tell you , the probability is that the total of the faces will be somewhere in the middle of the range of possible numbers - just as a pair of six sided die "like" to come up 7.

The trick is not to roll all 5 dice every time, but rather to come up with a scheme that most frequently rolls one or two and infrequently rolls all 5. Since the random number that is produced is always a total of the 5 dice, this produces a series of numbers that most frequently vary only slightly from one another while still permitting periodic large changes.

Voss's scheme (and ours) is to maintain a 5 bit "pinking counter" (our term) which is incremented each time we get ready to generate a new pink number. The new value of the pinking counter is compared to the old and only those "die" which correspond to bits in the counter which have changed are rolled.

The rest of the program is "overhead". As I mentioned in the beginning, PINK TUNES actually generates a 4 part harmony (provided that we supply it with harmonizing notes in the candidate list) and the program must keep track of how long each of the notes in the 4 parts is to play and allow for the updating of the candidate list and recognize a limited number of commands from the computer's keyboard.

The fully documented listing is the best place to go to see how it all works (it's in your best interests to understand it as fully as possible) and specific details and asides are covered in the boxes.

After entering the program and its data base (note part of the program is on page zero, part is on page one and the data base and working registers are on page zero), first save a copy on tape. If something goes crazy, you don't want to have to enter it all again.

Set up the synthesizer and start running the program starting at the hard start location of $0003. The data that you loaded is for the pentatonic scale composition that I mentioned in the opening paragraph and you should immediately hear the synthesizer producing the composition. It should go without saying that you will undoubtedly have to call the tuning function (control key #1) and tune the oscillators before it makes music.

You have the ability to change the candidate note list while the program is running simply by pressing keys on the keyboard, but bear in mind that the candidate list is 16 notes deep. As you enter a new note, the one that was entered "16 notes ago" disappears from the list. If any of the 16 notes are inharmonious, the program will periodicly produce discordant sequences.

With PINK TUNES running, three of the computer's control keys have meaning:

Key 0 "scrambles" the random number generator to produce a new tune. This is really only useful if you are in the cyclic mode (see box).

Holding key 1 provides a tuning function by causing all 4 outputs to produce a triggered middle C.

Touching key #2 initiates a muted shut-down of the synthesizer and branch back to the monitor, allowing changes in the memory locations described in the boxes.

After making changes using the monitor, always start the program running again from the soft start location $000B.

The program runs very nicely, but is experimental and not intended as a finished product. Skillful polishing should reduce its length by at least 15 - 20% and it would be nice to make changes in timing, etc. "on the fly" without having to shut down the synthesizer.

At the same time that the program is primarily "just for fun", don't dismiss it as trivial. It definitely produces 4 part harmonies and even those that are not directly useable in a composition can serve as inspirational lubrication to the gears of creativity. If you're involved in producing commercial jingles, this is a terriffic tool.

As you play with the program you will begin to get a feel for how various probabilities affect the composition and you're sure to learn some things about composition that you never knew before.

Finally, a very special thanks to Bob Yannes who sent me a listing of a similar program (PINK FREUD) which generates 4 part canons on a P4700/J. I haven't reviewed this program thoroughly yet, but knowing Bob it's sure to be neat. I'm sure that he wouldn't mind my sharing copies of the listing with anyone who sends a SASE.

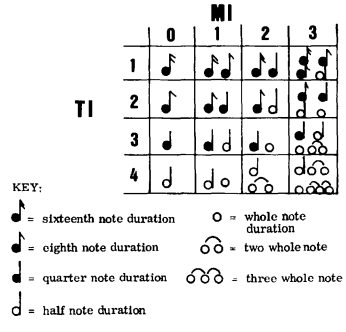
'Til next time, my best to all.

NOTE DURATIONS

Each of the 4 output channels has associated with it its own duration timer and two variables in the computer's memory which determine what characteristics the time values of the notes produced by that channel will have. In the interest of convenience, we'll name these two variables MASK and TIME; or, simply M and T.

We need to think of each of these variables as being composed of a high half-byte (hhb) and a low half-byte (lhb). The hex number $F3 (an arbitrary example) has an hhb of $F and an lhb of $3. This is necessary because the half-bytes determine two separate parameters.

The lower half-bytes of MASK and TIME (M1 and T1 respectively) interact to determine what time values are possible from a given channel. A channel can be restricted so that it produces only 1/16 notes or 1/16 and 1/8 notes or a wide variety of other possibilities as summarized in the table below:



Note that this is a partial table intended only to demonstrate the pattern.

Other combinations of M1 and T1 produce other possible time values. Some combinations not listed will produce undesirable results.

The high-half-bytes of MASK and TIME (Mh and Th) interact to determine the probability that the note being produced by that channel will be dotted (its duration extended by half of its actual value).

In actual practice, it is most convenient to set Mh to $F and regulate the probability using only Th. The influence of Th on the probabilities of a dotted note is illustrated below:

Th	Probability of dotted note
$8	one in two
$4	one in four
$2	one in eight
$1	one in sixteen
$0	zero

EXAMPLE: A channel which has MASK and TIME values of $F3 and $11 respectively will be capable of producing 1/16, 1/8, 1/4 and 1/2 notes with a one in sixteen probability of the note being dotted. A channel with M and T of $F0 and $01 will produce nothing but 1/16 notes, none of which will be dotted.

The page zero addresses of the MASK and TIME parameters for the four output channels are given below:

	CHANNEL A	B	C	D
MASK	$8F	$8E	$8D	$8C
TIME	$8B	$8A	$89	$88

TEMPO

By using the MUS-1 subroutine LOOK to gather data from the AGO keyboard, PINK TUNES follows our standard protocol of using the keyboard encoder clock rate as the system master clock. Analog control of tempo may be provided by varying this clock rate as has been mentioned in previous columns.

PINK also has a variable at zero page location $A9 which gives gross digital control of tempo. The recommended range of values for this variable are from $FF (far too fast) to $F0 (insanely slow).

GLIDE AND TRANSPOSE

PINK uses the MUS-1 QuASH drivers (NOTE) and therefore allows for both independent pitch transpositions of any and/or all 4 channels as well as providing a means of enabling or disabling glides.

Though not strictly true, it is most convenient to think of these variables as being divided into high half-byte and low half-byte with the hhb controlling glide ($8 turns the glide on, $0 turns it off) and the lhb determining transposition. For example, a channel which has this transposing variable set to $8C will have its glide turned on and be playing notes an octave higher than the actual note selected by PINK.

Here are the transposing variable addresses:

	CHANNEL A	B	C	D
TRANSPOSE	$CF	$CE	$CD	$CB

CYCLE CONTROL

The variable at zero page location $D3 controls the number of notes which will be played before the cycle repeats. Changing the contents of this location to $20 (for instance) will cause 4 bars of eighth notes to be played before the tune repeats. $40 would produce 8 bars of eighth notes.

Setting the contents of the location to $00 amounts to enabling a "free run" mode in which the patterns do not repeat (in practical terms).

If you want to get really fancy, you can change program location $188 from its current value of 85 (STA to the zero page) to EA (a NOP) and the result will be that on successive cycles the time values of notes will not change but the actual notes played will, producing a strong rhythmic tie from cycle to cycle. It also doesn't always work, sometimes a repeating loop will be entered anyway. Other times the duration of a tune will be 2 or more times as long as the actual cycle time.

To change a cyclic tune, touch control key 0.

DE-PINKING

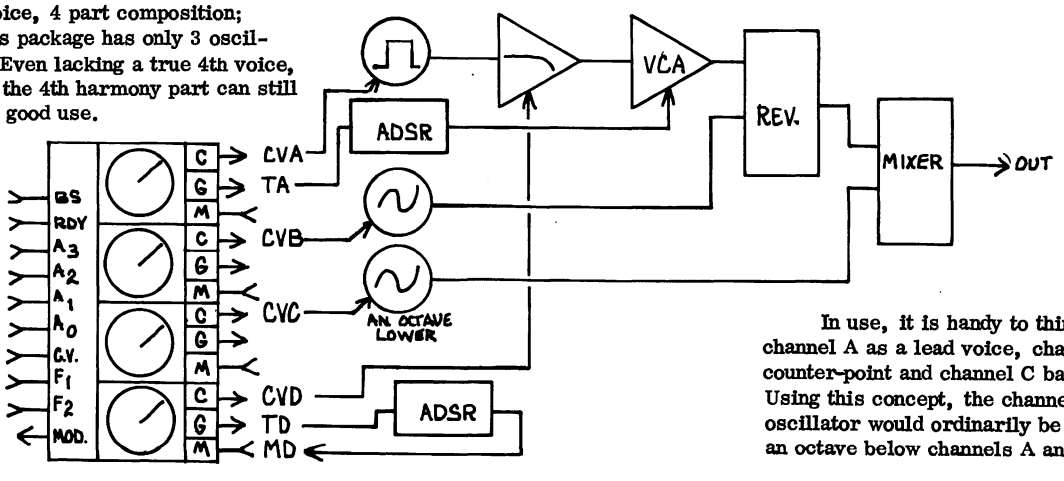
To get some feel for the effect that the Voss pink-ing algorithm has on the composition, you may want to change it slightly. There are a couple of easy ways that this can be done. By changing the current instruction at program location $11C from $45 (Exclusive-OR on the zero page) to $EA (a NOP), you slightly de-pink the note selector, making it somewhat more random. You may have to listen a while before you notice the difference, but there is one.

To completely eliminate the Voss algorithm make these substitutions beginning at location $118; A9 FF EA EA EA EA. This change is equivalent to rolling all 5 of our alleged 4 sided dice each time a new note is selected and will produce changes that even a tone deaf aborigine would recognize.

THE SYNTHESIZER

The module complement of a P4700/J is not large enough to do a true 4 voice, 4 part composition; since this package has only 3 oscillators. Even lacking a true 4th voice, however the 4th harmony part can still be put to good use.

Here is the most universal of the patches used during the development of PINK TUNES:



Note that the 4th harmony part (from channel D of the QuASH) is used to set the center frequency of the VCF.

In use, it is handy to think of channel A as a lead voice, channel B counter-point and channel C bass line. Using this concept, the channel C oscillator would ordinarily be tuned an octave below channels A and B.

THE CANDIDATE NOTES

Selection of the candidate notes that you give PINK TUNES and the order in which they're entered play a big part in the feel of the final composition. As an obvious example, the pseudo-pentatonic scale resulting from entering only accidentals (sharps and flats) tends to produce oriental sounding compositions.

The selection of notes is "pinked" on a compositional (rather than a per-channel) basis, which means that the 4 notes being played at any one time tend to cluster around a relatively short series of entries in the candidate table. The significance of this is that it allows statistical control of changes in key signature. For example, entering the candidate sequence C1, E1, G1, C2, E2, G2, C3, G2, A2, F2, D2, A1, F1, D1, F1, A1 will produce a composition that periodicly changes from the key of C to D minor.

It is important to remember that the candidate table will always contain 16 notes and in order to produce consistant harmonies, all 16 notes must be harmonious. Also remember that notes at the ends of the table (oldest and newest entries) have a lower probability of being played than the notes in the middle.

LOADING THE PROGRAM

NOTE THAT PINK TUNES CONSISTS OF THREE MAJOR SECTIONS: THE MAIN PROGRAM ON PAGE 0 OF MEMORY, SUBROUTINES ON PAGE 1, AND DATA BASE ON PAGE 0.

BEFORE ENTERING ANY PROGRAMMING, MAKE SURE THAT THE MONITOR STACK AND USER'S STACK ARE BOTH SET TO $FF (SO THAT THE STACK DOES NOT OVER-WRITE PROGRAMMING ON PAGE 1) AND THAT THE STATUS REGISTER IS SET TO $00 (TO INSURE THAT THE CPU IS WORKING IN THE HEXADECIMAL MODE) USING THESE ENTRY SEQUENCES:

```
0ED-DISP-FF-ENT (SETS MONITOR STACK)
0FE-DISP-FF-ENT-00-ENT (USER STACK
          AND STATUS REGISTER)
```

ALL OF THE FOLLOWING PROGRAMMING, DATA BASE, AND INITIALIZATION OF MUS-1 NEED BE DONE ONLY ONCE. THEY WILL SUBSEQUENTLY LOAD TO THE COMPUTER'S MEMORY FROM THE MASTER TAPE THAT YOU WILL GENERATE AT THE END OF THE LOADING PROCESS.

INITIALIZE THE MUS-1 VARIABLES CTRL ($0E8) AND ODLY ($0E9)

```
0E8-DISP-00-ENT-20-ENT
```

ENTER THE DATA BASE LISTED BELOW BEGINNING AT LOCATION $000 USING THIS ENTRY SEQUENCE:

```
000-DISP-02-ENT-04-ENT-01-ENT (ETC)
```

```
**********************************
          DATA BASE

   000:02 04 01 01 F2 F0 F3 F3
       5A 5D 5F 62 64 62 5F 5D
       5A 58 56 53 51 53 56 58
       00 00 00 00 00 00 00 00
       FA

**********************************
```

NEXT LOAD THE MAIN PROGRAM:

```
000-DISP-4C-ENT-C0-ENT-FF-ENT (ETC)
```

AND THE SUBROUTINES:

```
100-DISP-8A-ENT-48-ENT-A5-ENT (ETC)
```

BEFORE TRYING TO RUN THE PROGRAM SAVE IT ON TAPE FROM LOCATION $0 TO $1A0:

```
0-0-0-0-0-1-A-0-0-1-D-D-TAPE
```

BEGIN RUNNING THE PROGRAM FROM THE 'HARD START' LOCATION $3:

```
003-RUN
```

AFTER A SHORT (3 SECONDS OR SO) DELAY, THE PROGRAM WILL BEGIN PRODUCING THE COMPOSITION.

THE 'SOFT START' LOCATION IS $00B

```
                0010 :*******************************
                0020 :*                             *
                0030 :*         PINK TUNES          *
                0040 :*                             *
                0050 :*     A COMPOSING PROGRAM     *
                0060 :*    FOR FOUR PART HARMONIES  *
                0070 :*                             *
                0080 :*     BY JOHN S SIMONTON, JR  *
                0090 :*(C) 1978 PAIA ELECTRONICS, INC *
                0100 :*******************************
                0300 :
                0310 :
                0320 :FIRST ATTEND TO HOUSKEEPING--
                0360 :
000-  4C C0 FF   0370 BEG  JMP BRAK     :BREAK VECTOR
003-  20 21 0D   0380 STAR JSR INIT     :SET UP SYNTH
006-  AD 10 08   0390      LDA KBD      :INITIALIZE RANDOM
009-  85 D0      0400      STA *NTMP+01 :NUMBER GENERATOR
00B-  20 71 01   0410 LOOP JSR SET      :INIT PINK TUNES
00E-  20 2B 00   0420 LP0  JSR NOTE     :PLAY NOTES READ AGO
                0430 :
                0440 :CHECK FOR ADDITTIONS TO CANDIDATE
                0450 :NOTE TABLE
                0470 :
011-  A5 E7      0480 MAIN LDA *KTBL+08 :ANY KEYS DOWN?
013-  F0 02      0490      BEQ OUT1     :NO-CHECK FOR TIME OUT
015-  C5 EC      0500      CMP *TEMP    :YES-A NEW KEY?
017-  85 EC      0510 OUT1 STA *TEMP    :SAVE FOR NEXT TIME
019-  F0 0A      0520      BEQ OUT      :BRANCH IF SAME KEY
                0530 :
01B-  A2 10      0540      LDX 10       :IF NEW KEY SHIFT
01D-  B4 8F      0550 LP3  LDY *NBUF,X  :ALL 16 CANDIDATES
01F-  95 8F      0560      STA *NBUF,X  :DOWN BY ONE
021-  98         0570      TYA
022-  CA         0580      DEX
023-  D0 F8      0590      BNE LP3      :NOT DONE-LOOP
                0600 :
                0610 :NOW CHECK FOR CLOCK TIME OUT
                0620 :
025-  A5 BF      0630 OUT  LDA *CLCK    :GET MASTER CLOCK
027-  D0 1A      0640      BNE TEST     :AND IF TIMED OUT
029-  A5 A9      0650      LDA *TMPO    :SET TO TEMPO VALUE
02B-  85 BF      0660      STA *CLCK    :CALL SUB FOR NEW
02D-  20 53 01   0670      JSR ALOC     :NOTES (IF NEEDED)
030-  A5 A8      0680      LDA *LNTH    :GET CYCLE STATUS
032-  8D 20 08   0690      STA DISP     :SHOW IT AND IF ZERO
```

```
035-  F0 0C     0700        BEQ TEST     :CYCLE IS COMPLETE
037-  C6 A8     0710        DEC *LNTH    :IF NOT DONE, DCRMNT
039-  D0 08     0720        BNE TEST     :IF NOT ZERO NOWLEAVE
03B-  20 71 01  0730        JSR SET      :IF ZERO, REINIT
03E-  20 53 01  0732        JSR ALOC     :GET FIRST NOTES AND
041-  F0 CB     0735        BEQ LP0      :BRANCH ALWAYS TO PLAY
043-  20 00 0F  0740  TEST  JSR DECD     :GET A COMMAND
046-  D0 0C     0750        BNE TST2     :NOT ZERO,NEXT TEST
048-  A2 03     0755        LDX 03       :COMMAND 0,NEW TUNE
                0757                     SET POINTER/COUNTER
04A-  20 00 01  0760  TST1  JSR RNDM     :GET RANDOM NUMBER
04D-  95 CF     0762        STA *NTMP,X  :NEW INITIAL RANDOM
04F-  CA        0764        DEX          :POINT TO NEXT
050-  D0 F8     0766        BNE TST1     :NOT DONE - LOOP
052-  F0 B7     0770        BEQ LOOP     :BRANCH ALWAYS
054-  C9 01     0780  TST2  CMP 01       :COMMAND 1, TUNEING
056-  D0 0C     0790        BNE TST4     :NOT 1, TEST NEXT
058-  A2 04     0800        LDX 04       :4 OUTPUT BUFFERS
05A-  A9 5C     0810        LDA 5C       :PUT MIDDLE C IN ALL
05C-  9D DB 00  0820  TST3  STA NTBB,X   :OUTPUT BUFFERS
05F-  CA        0830        DEX
060-  D0 FA     0840        BNE TST3     :NOT DONE-LOOP
062-  F0 AA     0850        BEQ LP0      :BRANCH ALWAYS
064-  C9 02     0860  TST4  CMP 02       :COMMAND 2, STOP
066-  D0 A6     0870        BNE LP0      :NO COMMAND - LOOP
068-  20 71 01  0880        JSR SET      :CALL TO ZERO OUT-BUFFS
06B-  20 2B.0D  0890        JSR NOTE     :THEN MUTE SYNTHESIZER
06E-  00        0900        BRK          :AND RETURN TO PIEBUG
```

SUBROUTINES

```
                0220  :    RANDOM NUMBER GENERATOR
                0230  :
                0231  :ESSENTIALLY A 22 BIT LONG SHIFT
                0232  :REGISTER WITH EX-OR TAPS AT
                0233  :STAGES 22 AND 21 FED BACK TO
                0234  :INPUT.
                0235  :
100-  8A        0240  RNDM TXA           :SAVE X
101-  48        0250       PHA
102-  A5 A5     0260       LDA *NOIS+01 :LAST BYTE S/R
104-  0A        0270       ASL          :ALIGN BITS 22 &
105-  45 A5     0280       EOR *NOIS+01 :21 AND DO EX-OR
107-  0A        0290       ASL          :THEN SHIFT RE-
108-  0A        0300       ASL          :SULT TO CARRY
109-  0A        0310       ASL
10A-  A2 03     0320       LDX 03       :SET UP PNT/CNT
10C-  36 A4     0330  LP1  ROL *NOIS,X  :AND SHIFT 3 BYTE
10E-  CA        0340       DEX          :SHIFT REGISTER
10F-  D0 FB     0350       BNE LP1      :BY ONE BIT LEFT
111-  68        0360       PLA          :WHEN DONE RE-
112-  AA        0370       TAX          :STORE X REG.
113-  A5 A7     0380       LDA *NOIS+03 :AND LEAVE WITH
115-  60        0390       RTS          :WITH NO. IN ACC.
                0400  :
                0410  :        NEW NOTE
                0411  :
                0412  :TAKES CARE OF PICKING PINK NOTE
                0413  :FROM CANDIDATE NOTE TABLE AND
                0414  :CALCULATES AND UPDATES NOTE TIMERS
                0415  :NOTE THAT Y POINTS TO CHANNEL FOR
                0416  :UPDATE
                0420  :
116-  A2 05     0430  NWNT LDX 05       :SET UP PNT/CNT
118-  A5 EA     0440       LDA *OUTS    :GET COPY PINKING
11A-  C6 EA     0450       DEC *OUTS    :COUNTER,DEC ORIGINAL
11C-  45 EA     0470       EOR *OUTS    :PATTERN OF CHANGED
11E-  85 EB     0490       STA *OUTT    :BITS - SAVE CHANGES
120-  A9 00     0500       LDA 00       :PREPARE TO SUM DICE
122-  46 EB     0510  NW1  LSR *OUTT    :CHECK FOR CHANGED
124-  90 0A     0520       BCC NW2      :BIT - IF CHANGED,
126-  48        0530       PHA          :SAVE CURRENT TOTAL
127-  20 00 01  0540       JSR RND      :GET RANDOM NUMBER
12A-  29 03     0550       AND 03       :MAKE RANGE 0 TO 3
12C-  95 9F     0560       STA *RAND,X  :SAVE VALUE FOR NEXT
12E-  68        0570       PLA          :RECOVER TOTAL
12F-  18        0580       CLC          :PREPARE ADDITTION
130-  75 9F     0590  NW2  ADC *RAND,X  :ADD VALUE OF DIE
132-  CA        0600       DEX          :POINT TO NEXT
133-  D0 ED     0610       BNE NW1      :LOOP IF NOT DONE
135-  AA        0620       TAX          :USE TOTAL AS POINTER
136-  B5 90     0630       LDA *NBUF,X  :GET CANDIDATE
138-  F0 03     0640       BEQ DURA     :ZERO, DO NOT CHANGE
13A-  99 BF 00  0650       STA NTB7,Y   :PLACE IN TEMP BUFFER
13D-  A5 A5     0660  DURA LDA *NOIS+01 :A CHEAP RANDOM NO.
13F-  18        0670       CLC          :PREPARE
140-  39 8B 00  0680       AND MASK,Y   :MASK DURATION VAL.
143-  79 87 00  0690       ADC TIME,Y   :ADD MINIMUM VAL.
146-  29 0F     0700       AND 0F       :AND MASK RESULT
148-  AA        0710       TAX          :USE AS COUNTER AND
149-  A9 01     0720       LDA 01       :DO DURATIONS AS
14B-  2A        0730  NT2  ROL          :POWERS OF 2. CARRY
14C-  CA        0740       DEX          :SET DOTS NOTE
14D-  D0 FC     0750       BNE NT2      :NOT DONE - LOOP
14F-  99 C3 00  0760       STA NTBB,Y   :PUT RESULT IN NOTES
152-  60        0770       RTS          :TIMER AND RETURN
                0780  :
                0790  :ALLOCATION 0151
                0791  :
                0792  :SEES IF NEW NOTES ARE NEED AND IF
                0793  :SO GETS THEM.  ALSO CLEARS TRIGGER
                0794  :OF NOTE OUTPUT ONCE IT IS PLAYED.
                0800  :
153-  A2 04     0810  ALOC LDX 04       :DO 4 NOTE CHANNELS
155-  D6 C3     0820  LP6  DEC *NTBB,X  :DECREMENT NOTE TIMER
157-  D0 07     0830       BNE LP5      :AND IF TIME OUT
159-  8A        0840       TXA          :TRANSFER X REG. TO
15A-  A8        0850       TAY          :TO Y
15B-  20 16 01  0860       JSR NEW      :AND GET NEW NOTE
15E-  98        0870       TYA          :AND DURATION AND
15F-  AA        0880       TAX          :RESTORE X
160-  CA        0890  LP5  DEX          :DECREMENT COUNTER
161-  D0 F2     0900       BNE LP6      :IF NOT DONE - LOOP
163-  A2 04     0920       LDX 04       :AGAIN, FOUR CHANNELS
165-  B5 BF     0930  AL1  LDA *NTB7,X  :GET NOTE FROM TEMP
167-  95 DB     0940       STA *NTBB,X  :BUFFER, SAVE IN OUT
169-  29 3F     0950       AND 3F       :BUFFER, CLEAR FLAG
16B-  95 BF     0960       STA *NTB7,X  :PUT BACK IN TEMP.
16D-  CA        0970       DEX          :POINT TO NEXT
16E-  D0 F5     0980       BNE AL1      :NOT DONE - LOOP
170-  60        0990       RTS          :DONE, RETURN
                1000  :
                1010  :            SET
                1011  :
                1012  :PREPARES KNOWN STARTING POINT FOR
                1013  :CYCLIC TUNES.
                1014  :
                1020  :
171-  A9 00     1030  SET  LDA 00       :TO ZERO THINGS WITH
173-  A0 01     1040       LDY 01       :PRESET FOR NOTE CNTRS
175-  A2 04     1050       LDX 04       :DO 4 CHANNELS
177-  95 DB     1060  LP10 STA *NTBB,X  :ZERO OUT-BUFFERS
179-  95 A0     1070       STA *RND0,X  :ZERO 4 DICE
17B-  94 C3     1080       STY *NTBB,X  :PRESET NOTE TIMERS
17D-  48        1090       PHA          :SAVE THE ZERO
17E-  B5 CF     1100       LDA *NTMP,X  :SET UP RNDM'S S/R
180-  95 A4     1110       STA *NOIS,X  :AND CYCLE COUNTER
182-  68        1120       PLA          :RECOVER ZERO
183-  CA        1130       DEX          :POINT TO NEXT
184-  D0 F1     1140       BNE LP10     :NOT DONE - LOOP
186-  85 A0     1150       STA *RND0    :ZERO 5TH DIE
188-  85 EA     1160       STA *OUTS    :ZERO PINKING COUNTER
18A-  60        1170       RTS          :AND RETURN
                1180  :
                1190  END  EN
```

SEQUE 1.0

UNIVERSAL MONOTONIC SEQUENCER

REAL TIME MODES

Now we're going to start a long discussion of sequencers.

It's going to be long because there is no single kind of sequencer that's best in every situation. Some will do better on stage and others will be more at home in a studio setting. Polyphonic sequencers should at times be structured for storing and reproducing chord sequences while at other times each channel should be treated as a separate voice. The only really workable solution is to come up with an entire "family" of sequencers.

The common limitation of all programming devices currently available is that none of them can offer this kind of versatility. But, this is an area where the system that we've developed, with its ability to accept a wide variety of personality endowing programs, will really come into its own. If we need a studio sequencer (with click track synchronization and full score editing features, etc.) we can load that program; when a chord sequencer is required, that software can be loaded.

With few exceptions, these programs will all be "complete" in that once they are running, the system loses any "computer personality" that it may have had. All of the features that the program offers will be available with one or two touches of the "command" (computer) keyboard. You can forget that the computer's there because its control keys are dedicated exclusively to functions assigned them by the program. "This key makes it play - this key makes it play faster." Easy.

To illustrate these points, we'll begin with a program called SEQUE 1.0, a monotonic sequencer written to run on a PAIA P-4700/C or its equivalent. It can also be easily patched to run on a P-4700/J as outlined in the box.

SEQUE 1.0 is an acceptable "general purpose" sequencer (acceptable from the standpoint of our new perspective - in terms of the alternatives that are available it is the most sophisticated sequencer ever produced). It has some features tailored for live performance and others that are primarily for studio use. The program listing and some additional notes appear in following pages.

COMMAND KEYS

When SEQUE 1.0 is running, the command keys should be thought of as being labeled like this:

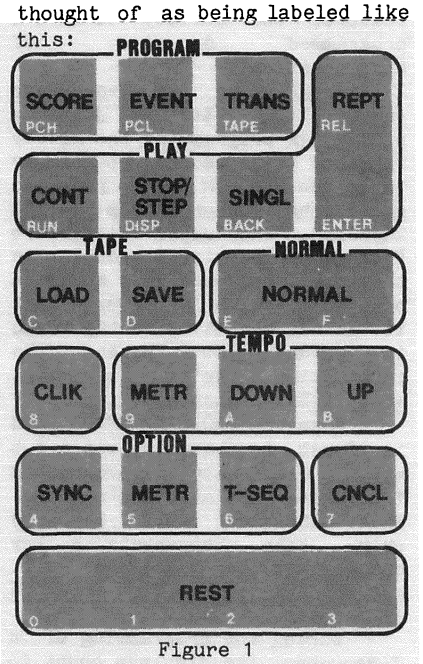


Figure 1

Undoubtedly, some of the designations on the keys still seem a little on the cryptic side. Let's look at function and begin by pointing out some of the ways that SEQUE 1.0 is different from what you're accustomed to.

PROGRAMMING A SEQUENCE

The first way that it's different is that you don't program it with knobs, you simply enter the note sequence from the AGO keyboard. More specifically the first operating mode that we'll examine is a completely "real time" performance mode. You simply touch the "PROGRAM SCORE" key and start playing. Except for the fact that we will be able to do much magic, the result is the same as if there were a tape recorder somewhere recording what you're playing. Whatever tempo you play in, including subtle timing nuances, are faithfully captured by SEQUE 1.0 and stored in the computer memory. When you reach the point at which you want the sequence to repeat, touch REPEAT PLAY and it all comes back.

PLAYING THE SEQUENCE

Since this is a real time mode, the timing of punching up REPEAT PLAY is important. If you were storing a repeating bass line, for example, you would play the single figure that characterizes the bass line and then, at the exact point (and on the beat) where the first note of the figure was to be repeated, touch REPEAT.

There are other sequencers beginning to appear that operate this way, and if real music was played with droning bass lines that repeat unchanged, endlessly,

they would be perfectly adequate. And the music would be perfectly boring.

Not that real music doesn't frequently have the characteristic of a repeating bass figure, it does, but it's also made to sound different by transposing the figure into different keys to follow key changes in the composition. While this fact seems to have been largely ignored by sequencer manufacturers, we don't have to settle for that.

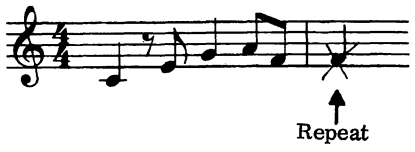
TRANSPOSING

SEQUE 1.0 has a variety of provisions for transposing the programmed sequence. The simplest of these is that while in playback mode it can accept information on key changes directly from the AGO keyboard. A little explanation.

Since we obviously want to be able to transpose both up and down in pitch, we need to decide that some arbitrary key represents no transposition (play the sequence as programmed). SEQUE 1.0 assumes that the 2nd C on the keyboard is the "0 transpose" key. keys up-scale and down-scale from this one, then, represent transpositions up and down scale respectively. press the C# above the 2nd C , and the entire sequence plays with each note a semi-tone higher than was originally programmed. Press the F below the 2nd C and then each note plays a fifth lower.

As an example of this, suppose that we were going to want to play a walking bass line as shown in figure 2.

Because of the things we've talked about already, it should be relatively obvious that we only need to really "play" this much of the entire bass line:



(NOTE: Do not hit this note! Hit repeat at exactly the time you would have played it.)

Figure 3

because from then on it simply repeats, transposed into different keys. As the riff from figure 3 plays, we can extend it out to the entire bass line simply by pressing keys on the AGO keyboard to perform the appropriate transpositions at the proper time. Like that shown in figure 4. Pretty exciting. And we really haven't even started yet.

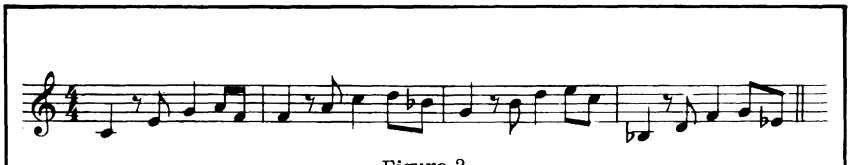


Figure 2

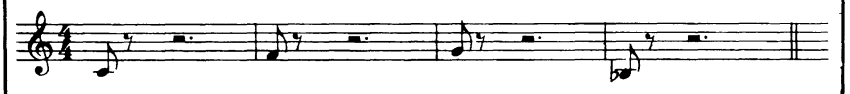


Figure 4

THE TRANSPOSE SEQUENCE

While being able to transpose the programmed sequence with real time keyboard entries will be plenty useful again and again, there are also going to be times when it will be at best a pain in the neck. You'll be busy doing other things. For these times, SEQUE 1.0 offers another feature, the ability to save a programmed sequence of transpositions.

Programming the T-sequence (as we'll call it) is just as simple as programming the melody sequence (M-sequence), you simply touch the PROGRAM TRANS pad and enter the sequence from the AGO keyboard. The major difference from a programming standpoint is that the T-sequence is a sequence of events, which is to say that it is not sensitive to the tempo in which you enter the information. We'll talk more about EVENT sequences later.

When the PROGRAM TRANS pad is first touched, it wipes out any previously programmed T-sequence and starts a new one. Each subsequent AGO keyboard entry then represents a key change that the M-sequence will go through at the point at which it repeats.

During the programming of a T-sequence, the displays count to show where we are in the sequence, and the note corresponding to the transposition will play while the key is held down. When the key is released, the note stops completely, so that there is no possibility of confusing this programming mode with others.

On playback, the M-sequence will be played completely through, transposed to the key signature corresponding to the first T-sequence entry; then completely through transposed by the second T-sequence entry, then the third, etc. When the end of the T-sequence is reached, the whole thing starts over again with the first note and the first T-sequence entry. To go back to our walking bass line for a moment, the T-sequence would program like this:

Figure 5

In the terms which we will find most useful, enabling the automatic transpose is an OPTION which may be selected along with one or more of the major operating MODES. If we want to assert the T-sequence option during playback we do so by touching the T-seq. OPTION key. To stop the T-sequence and revert to the manual entry of transpositions, simply touch the OPTION CANCEL pad.

It is important to note that canceling the T-seq. option simply keeps the system from invoking the T-sequence, and does not in any way alter the sequence as stored. You can turn the option on and off as many times during a set as desired.

And still there's more.

SINGLE PLAY

There will be times when we don't want the sequence to repeat endlessly, but simply to play one time through and stop. A SINGLE PLAY MODE.

An important difference between the two modes is that whereas REPEAT begins playing the sequence as soon as it is touched, SINGLE PLAY waits for an AGO key to be pressed and then plays.

The T-sequence option may also be asserted in the SINGLE PLAY MODE, but it has been my experience that it's not tremendously useful. Much more useful is to have the T-seq. option cancelled (which selects the AGO keyboard as the transposition source), so that pressing an AGO key not only starts the sequence playing, but causes it to play in the key selected.

Releasing the key which initiated the sequence will not cause it to stop (once started it always plays to the end), but pressing a different key in the middle of the sequence will immediately transpose it to a new key signature.

TEMPO KEYS

The function of the TEMPO UP and TEMPO DOWN keys is just what you would expect. Touch TEMPO UP and the tempo of the sequence being played doubles. Touch it again and the tempo doubles again. Touch TEMPO DOWN and the tempo rate is divided in half.

If not over-used, these two keys will increase and decrease tempo while still keeping relative timing of notes unchanged; however, raising the tempo too high will cause some timing information to be lost and will cause the notes to be "jammed" together so that synchopation will change. Beware and be aware that this fact has special effects implications - there may be times when you want to do just this.

TAPE SAVES AND LOADS

The TAPE pads control a couple of operating modes which should also be useful. TAPE SAVE causes the M-sequence and T-sequence information currently in the computer's memory to be dumped to magnetic tape. when you come up with a "keeper" start your recorder going (recording) and touch TAPE SAVE. After a short leader and synchronizing tone is generated, the displays will start to count and within a few seconds your complete composition will be stored as data on the tape (a hint - always save things twice)

Loading a composition that was previously saved on tape consists of playing the tape and touching the TAPE LOAD command pad. As with the saving operation, the displays count as the data transfers from tape to memory. If, after loading a tape, you punch up PLAY MODE and nothing happens, it means that the load was unsuccessful. Try again with the second copy (and review the "tape selection" section of PAIA's CS-87 POT SHOT manual).

NORMAL MODE

NORMAL is simultaneously the most straightforward and ubiquitous of all the operating modes. NORMAL is nothing more than a normal monotonic synthesizer function, the important point is that asserting this mode of operation does not alter previously programmed M or T sequences. It simply ignores them as long as this mode is selected. at any time you can punch-up SINGLE or REPEAT PLAY and do that magic and with a touch of the NORMAL pad be back to plain synthesizer.

SUBTLETIES AND TRICKS

It seems to me that a sequencer for use on stage should have two major design goals: it should be easy to program and operate (which SEQUE 1.0 certainly is) and it should enable the user to do a better job of the thing he's there to do - put on a show. As theatrical a show as possible. SEQUE 1.0 has several of these "show" features.

The ability to shift back and forth between the various modes of operation (and specifically the availability of the NORMAL mode which doesn't mess up programmed sequences) is definitely one of these.

Others are less obvious, for example:

When you have the T-sequence option selected (so that transpositions come from their programmed sequence) and you go directly from the PROGRAM SCORE mode to REPEAT PLAY without first asserting another operating mode, the first entry of the T-sequence will be skipped and the melody sequence will begin playing immediately transposed by the second entry in the Transpose Sequence.

Why?

Because, when you entered the characteristic sequence it was equivalent to its being played the first time through (which would have been done using the first T-sequence entry). When you hit REPEAT PLAY and the computer takes over, it is in effect playing the sequence the second time - which should be done in the key of the second T-sequence entry.

The major application here is to allow you to enter (during set-up and tuning) a T-sequence for the number that you are going to be doing and then enter the actual sequenced figure extemporaneously. We all know how great it is when the magic is working and everybody's really cooking. This feature allows your automation equipment to tap into that energy and the innovation that frequently results from it.

If for some reason you don't want to skip the first T-sequence entry, you simply terminate the PROGRAM SCORE mode with a command other than REPEAT PLAY (NORMAL, for instance; or SINGLE PLAY), then punch into REPEAT PLAY. Remember always, though, that the termination of PROGRAM SCORE mode must be done "in tempo" if the timing of the playback is to be correct.

Here's another special application:

In most cases, the M-sequence is reserved for the melody, but the UP TEMPO command allows you to enter some short riff (live, yet) then speed the sequence up to the point that it has the effect of being a "voice" of its own. By then punching into SINGLE PLAY mode, the sequence can then be used as you would a single note, which you "play" by transposing it. Naturally, the T-seq. option should be cancelled for this.

And another:

REPEAT PLAY mode always starts the M and T sequence from the beginning, making it any easy matter to use the first few bars of the sequence again and again, for introductions, bridges, and special effects.

STUDIO MODES

Now we turn our attention to the studio-oriented options offered by this "universal" monotonic sequencer program.

Some of the distinctions between stage and studio use are somewhat arbitrary.

For example:

EVENT PROGRAM

The real-time SCORE melody programming mode that we examined in the first section of this piece can obviously be used in a recording studio as well as it can on stage, providing that you're interested in recording only those things that are within the limits of your physical abilities. But the real promise of a small studio (or a big one, for that matter) is that it allows us to produce music that we don't have the chops to do in real time. After all, not everyone has the hours per day that it takes to gain physical mastery of a keyboard - but that doesn't mean that we don't have valid musical ideas, only that we need a little help in expressing them.

If a recording studio is a single thing, it's a time machine that allows days or weeks of work to be compressed into a few minutes of music. One of the programming modes that we have available (EVENT) is specifically designed to operate in this type of time-compression environment. In this mode we enter the music not so much as a melody, but as a series of notes and rests. A series of events which, when reproduced by the computer, turn out to be a melody (maybe).

There is of course nothing new about this mode of operation, this is the way sequencers have always worked. About the only new part is that instead of entering the events as positions of a knob or a series of numbers, we have an AGO keyboard on which to program.

Touching the command keyboard's PROGRAM EVENT pad puts us in this programming mode. (See Figure 1.) Melody lines are entered much as they were with the SCORE mode, except that the computer is no longer watching for how long we hold a key down or how rapidly the notes are played. It is now only interested in whether a key is up or down.

One of the major implications of this is that notes in the melody are "jammed" together in time, and on playback will come out exactly equally spaced, one note per beat. While this is OK in some cases, as a general rule it is unacceptable; because it is unacceptable, we have a REST pad on the command keyboard. The REST pad provides for syncopation. It is a means of "extending" an event so that it takes more than a single beat.

If you're familiar with the operation of the rest key on something like PAIA's Programmable Drum Set, you already have a good idea what's going on, but there still are some surprises here.

Your first thought may be that when you press and release a key on the AGO keyboard, that constitutes an event. Actually, it's two events as far as SEQUE 1.0 is concerned - the first when the key was pressed and the next when it was released. It's important to keep in mind that the REST pad can extend either of these events.

For example, this simple phrase:

Figure 6

would be entered from the keyboard by pressing F and releasing, press A and release, press C, release, press D, release, press F and while holding the F key down, hit the REST block on the keypad, release the F key, tap the REST block, play A, touch the rest block before letting up the A key, release the key, and hit the rest block once more. The measure is now completely entered, and may be played by using the REPEAT or SINGLE keys as described last time. Note particularly that on the fifth note (the second F) where we wanted to extend the note to a full beat, the REST pad had to be touched twice; once to extend the "key down" event and again to extend the "key up" event.

At first, having to enter two RESTs when we actually want to extend a note for a single beat may seem a pain in the neck (undeniably, it is) but the slight inconvenience buys us a number of things. For example, the ability to slur notes.

In the above example, the D could have been slurred into the F by first touching the REST pad before releasing the D key. This will lengthen the note to occupy the time normally used when the key is released. Then press the F key before releasing the D. This will cause the D to be entered in the next time slot without any articulation (triggering). Now, while holding the F key, touch the REST pad to lengthen it to a quarter note as covered earlier. After releasing the key, enter the additional REST required and proceed as usual.

Having each REST pad activation correspond to a "half" event (kind of) also allows us to produce dotted notes as the exceptions that they are rather than having to make specific tempo provisions for them which must be carried over to all notes in the sequence.

It is also possible to generate articulation changes whenever a note is extended beyond a basic "dual" event. If, for example, you are generating a series of notes where each note uses a key depression plus a REST and a key release plus a REST (four events), theses notes can be performed in three different manners. If entered as listed above, the note has equal time allotted for note performance and release. For a staccato style, the note could be entered with a key depression, release, and then two RESTs. For legato styles, the two RESTs could be entered while the key is held down, yielding three "on" events and one "off" event. Each of the above would occupy the same execution time during playback, but would reflect the different articulation styles.

Once the melody is in the computer's memory, it makes no difference whether it got there with SCORE or EVENT programming modes as far as the playback and options are concerned. All of

these features (real time or programmed transpositions, single or repeat play, tempo up and down, and tape saves or loads, etc.) work the same.

CLICK TRACK SYNCH

Even more powerful in the studio than the EVENT programming mode are the features added by two other command pads; CLIK and (in the option box) SYNC. These provide a means of synchronizing multiple tracks of sequencer operation.

Once you start using a sequencer for recording, you begin to find more and more places where it can be used to relieve a lot of tedium. The problem in the past has been that it is, for all practical purposes, impossible to manually synchronize a sequencer to a track that's already on tape. Even slight differences in tempo soon build up to an intolerable variation in when a note is supposed to happen and when it actually does happen. Maybe there are people who could manually twiddle a tempo knob and keep things locked together, but that's a hassle.

Most of us are familiar with the classical "click track" approach in which a metronome-like "tick" is recorded on one track of a tape so live musicians can easily maintain the tempo of the original work in over-dubs. Our CLIK and SYNC command pads are simply this old concept extended into the realm of automation.

Touching the CLIK pad causes SEQUE 1.0 to begin producing a very rapid series of "clicks" that are machine readable and represent a standard clock rate which the SYNC option can read and synchronize to. The click appears at the normal cassette output jack (where programs, etc., that are to be saved to tape come from) and when using this option, this output is tied to one of the channels of the tape recorder on which you're recording your audio tracks.

To use the click track option, the tape that you will be recording and mixing your audio onto must always be prepared first; you can't record a lead part and then come back and lay down the click, it won't work like that. Before doing anything else, connect the 8700's cassette output to the input of one track of your recorder, start the tape rolling in record mode, and after allowing a comfortable quiet leader, punch the CLIK pad. Allow the tape to run much longer than you think you'll ever need for what you're going to be recording, one thing you don't want to do is run out of click in the middle of things.

Synchronizing to the click track is simply a matter of connecting the output of the tape channel that contains the click to the normal cassette input jack of the computer, but note that some juggling of the record and playback levels of this channel may be necessary for the computer to properly write and read the channel. In many cases, unless your recorder is capable of providing very high outputs (similar to the earphone levels from the cassette recorders which the computer was designed to work with), you may need to use a small external amp to provide the extra gain and current drive required. If your SYNC fails to respond, try using the earphone jack signal usually provided on multi-track recorders. If this doesn't provide enough power, try using a small portale practice amp (such as a Pygmy or Pignose) whose earphone output should adequately drive the cassette input jack of the computer.

Assuming that you have some rhythm sequence (ordinarily the first laid down) in the computer memory and that you're getting ready to record it as audio, proceed by first punching into the T-SEQ option (if you plan to use it) then touch the SYNC control pad. Roll the tape with the click track channel set to playback and the audio going to one of the other tracks which is naturally in record mode. Before the quiet leader ends, touch the REPT/PLAY command pad and hold it. When the click track starts, so will the sequence. When enough of the track is laid down, terminate the play mode by touching the NORMAL pad.

It is necessary to select the SYNC OPTION last in the above sequence of events because once this option is asserted, a click track must be coming in on the cassette port for the computer to recognize any further commands. If you find yourself with a "dead" computer caused by CLIK being selected with no click track present, you can either run a tape which has a click track or reset the computer and run the program again.

In situations where the sequence is not to be played from the first down-beat, the SYNC OPTION should be enabled before rolling the tape and REPT/PLAY punched in when the time comes for the sequence to start.

A little constructive play will go a long way toward familiarizing you with the capabilities of this powerful option. Here are some we haven't mentioned yet:

You have probably already noticed the somewhat cryptic METR designations that appear in both the OPTION and TEMPO control boxes. And probably you've figured out that it means metronome (a handy thing in any studio). But this is kind of a super metronome because not only does it have a "pendulum" (which shows in the computer's twin displays) and an audible click (which you hear from the beeper) but it also provides an electrical output in the form of a short positive going pulse that appears as D7 of the D/A output channel (which in turn shows up on the Flag 2 pin jack of the D/A's front panel). This pulse is enormously useful in synchronizing external devices (a Programmable Drum Set, for example).

Since both the SYNC and METR options may be asserted at the same time, the external device can be synched to a pre-recorded audio track.

The METR pad in the TEMPO control box is obviously the tempo control for the metronome. Like the other tempo controls that we looked at last time, this one works in octaves. Each time the pad is touched the metronome tempo doubles until the maximum rate is reached, then the next touch causes the tempo to "fold back" to the minimum rate.

It may be somewhat out of sequence (?) to mention here that the tempo of the metronome is the tempo at which sequences stored in EVENT mode will play back, though of course, the TEMPO UP and DOWN command pads will also alter the tempo of the sequence once saved, as outlined last time.

Another point - When electrically synchronizing things to the click track, the METR TEMPO can still be varied to accomodate different timings, and since it operates by octaves the

integrity of the timing will be preserved.

And a hint - the metronome "beep" can also be recorded on tape to provide a "human readable" click track (though it must be saved on a different track than the CLIK).

The only other command pads that we've added are STOP/STEP (a means of stopping the sequence without "forgetting" where we were as well as single stepping through the sequence) and CONT (continue) which allows us to pick up from the point where we STOPped. This feature can provide easy introductions to songs. STOP/STEP through the piece until you reach the REST just prior to the point where the introduction should start. When the CONTINUE pad is touched, the introduction will play, leading into the repeating sequence.

SEQUE 1.0
COMMAND SUMMARY

PROGRAM

SCORE - Saves melody sequence in real time.

EVENT Saves melody sequence as regularly spaced events.

TRANSPOSE - Saves transpose sequence as events.

PLAY

REPEAT - Plays sequence from beginning, cycles until stopped.

SINGLE - Waits for key on AGO then plays sequence from the beginning. Stops at end of melody.

STOP/STEP - Allows stops or pauses during playback.

CONTINUE - Starts melody playback from where you are in memory.

TAPE

SAVE - Dumps current Melody and Transpose sequences to mag. tape.

LOAD - Loads M & T sequences from tape.

OPTIONS

TABLE - Selects transpose sequence table as source of transpositions (otherwise AGO is source).

METRONOME - Initiates visual metronome display and a "beep".

SYNC. - Shuts down internal timing and accepts prerecorded click-track for timing information.

CANCEL - Turns all selected options off.

TEMPO

UP - Doubles tempo of melody sequence.

DOWN - Halves tempo of melody sequence.

METRONOME - Doubles speed of metronome display and "beep"

MISC

NORMAL - The "normal synthesizer" mode. Does not alter stored sequences.

LOADING SEQUE 1.0

LOADING FROM TAPE

Because part of the SEQUE 1.0 program is held on the same page that the 6502 processor uses as a stack register, some very slight preparation is required before the SEQUE 1.0 tape can be loaded. Specificly the stack pointer which is used by the PIEBUG monitor program and the stack pointer associated with the applications program must be set to assure that the stack will not over-write the program. And, as long as we are setting the stack pointer, the status register may as well be set to a known state.

These objectives may be met by these keyboard manipulations:

0-0-E-D-DISP-F-F-ENT this sets the monitor stack

0-0-F-E-DISP-F-F-ENT-0-0-ENT this sets the user stack and status register.

On the tape supplied, SEQUE 1.0 is saved with the identifier 01; and should be loaded to memory from location $0000 to $0280 using this entry sequence:

0-0-0-0-0-2-8-0-0-1-1-1-TAPE

The program is saved in triplicate so if the first copy won't load for some reason you can always try for the next. All copies have the same identifier 01. If you experience continued difficulties in loading, refer to the POT-SHOT cassette interface manual.

HAND LOADING

If you are absolutely unable to load the program from this cassette, return it for a replacement. Since transit time back and forth may present unbearable delays, you may consider hand-loading the program and dumping your own tape (which goes a long way toward eliminating any problems caused by differences in tape recorders). To assist you should this solution become desirable, a hexadecimal dump of SEQUE 1.0 is provided below. NOTE that even if you hand load the program, the stack and status register setting manipulations outlined above should be performed before you start loading the program.

With the stack pointers and status register set, the program may be hand loaded as outlines in the various 8700 manuals:

First the programming on page 0:

0-0-0-DISP-A-9-ENT-0-0-ENT-8-5-ENT- (etc.)

Then page 1:

1-0-0-DISP-8-5-ENT-1-0-ENT-8-5-ENT- (etc.)

And finally page 2:

2-0-0-DISP-8-D-ENT-0-6-ENT-1-2-ENT- (etc.)

Note that none of these blocks go all the way to the end of the page.

When done loading, immediately save the program to tape from $0000 to $0280. Do this before running the program to avoid the unpleasant experience of having some incorrectly copied code wipe out the program. Next verify the program by stepping through it and comparing memory contents to the hex dump. Finally, when you're sure that it's entered correctly and have a copy on tape just in case, run it.

RUNNING THE PROGRAM

Location 0 is both the hard start and the soft start location for this program.

0-0-0-0-RUN.

If the program crashes (as perhaps when SYNC is selected with no synchronizing tape input) it may be re-started from this same location without losing any melody or transposing sequences that had been saved to that point. Re-starting from this location will cancel any options which may have been selected and will cause SEQUE 1.0 to come up in its NORMAL mode.

RUNNING SEQUE 1.0 ON A P-4700/J

SEQUE 1.0 may easily be modified to run on a polyphonic system (though it will still be a monotonic sequencer) simply by changing the address of the output port which appears at SEQUE 1.0 locations $14B & $14C. Changing this address to $09FF will cause the output to appear at QuASH channel #1. This may be accomplished as follows:

1-41B-DISP-F-F-ENT-0-9-ENT

If you make this change, you should also save the altered program to tape.

A fully documented assembler listing of SEQUE 1.0 starts on the following page.

SEQUE 1.0 HEXADECIMAL DUMP

PAGE 0

```
000- A9 00 85 E2 A9 0C 8D 7B
008- 11 4C 18 11 B0 05 85 E6
010- 8D 20 08 A5 EC D0 04 A5
018- EB 29 3F 85 EB 60 B0 0A
020- 85 E6 85 EB 85 E7 A9 00
028- 85 E2 A6 E7 8E 20 08 A5
030- EC F0 06 C5 EB F0 05 E6
038- E7 9D C8 02 85 EB 60 20
040- 84 11 E6 E5 60 38 20 AC
048- 11 AD 14 11 C5 E3 D0 02
050- E6 E9 24 E2 30 0A A5 EC
058- F0 02 85 E4 A5 E4 85 E6
060- E6 E5 60 90 04 A5 EC D0
068- DD 20 46 10 A5 EA D0 0B
070- A9 00 85 E5 A6 E8 BD 01
078- 03 85 EB 60 A9 7E D0 02
080- A9 3E 4C 00 12 18 A5 E5
088- 65 E1 85 E5 A5 E3 8D 7B
090- 11 60 B0 0E A9 FF 85 E5
098- 20 B6 11 8E 20 08 A9 00
0A0- 85 E6 60 B0 02 85 E5 20
0A8- 84 11 A5 E5 D0 05 18 65
0B0- E1 85 E5 60 85 E9 A5 E2
0B8- 09 80 D0 0E A5 E2 09 40
0C0- D0 08 A5 E2 09 01 D0 02
0C8- A9 00 85 E2 4C 0F 12 18
0D0- 20 25 1E 60 4C 54 12 4C
0D8- 20 12 4C 33 12 FF FF 00
0E0- 00 04 00 00 00 00 00 00
0E8- 00 00 00 00 00 FF FF FF
```

PAGE 1

```
100- 85 10 85 85 C2 BC B4 C8
108- CF D4 80 7C DA D7 0C 0C
110- 45 92 63 46 3F A3 1E 46
118- A5 E2 48 0A 10 22 C6 DF
120- 10 1E A6 E1 CA 86 DF A9
128- 80 AA 18 65 E0 85 E0 10
130- 0C A5 EB 09 80 85 EB 18
138- 20 25 1E A2 08 8E 20 08
140- A5 E6 F0 03 18 69 A4 18
148- 65 EB 8D 40 08 68 6A 90
150- 06 20 49 1E 4C 6D 11 2C
158- 10 08 10 FB AD 10 08 30
160- FB 2C 10 08 30 05 50 F9
168- AD 10 08 85 EC 20 00 1F
170- B0 06 B9 00 11 8D 7B 11
178- A9 00 20 03 00 AD 7B 11
180- 85 E3 D0 94 B0 09 8D 01
188- 03 85 E8 85 E6 85 EB A5
190- E5 A6 E8 9D 00 03 20 13
198- 10 29 7F DD 01 03 F0 0B
1A0- E8 E8 86 E8 9D 01 03 A9
1A8- 00 85 E5 60 B0 08 85 E4
1B0- 85 E9 85 EA 85 E5 A5 E5
1B8- A4 E9 A6 EA DD 02 03 90
1C0- 15 A9 00 85 E5 E8 E8 86
1C8- EA E4 E8 D0 09 C8 C4 E7
1D0- B0 DE 84 E9 D0 DC BD 03
1D8- 03 85 EB B9 C0 02 85 E6
1E0- 60 FF 00 FF 03 C0 02 C0
1E8- 02 FF FF FF FF FF FF FF
```

PAGE 2

```
200- 8D 06 12 A2 00 18 7E 02
208- 03 E8 E8 E4 E8 D0 F6 A5
210- E3 8D 7B 11 60 A2 07 BD
218- E1 11 95 F0 CA D0 F8 60
220- 20 15 12 A5 E8 8D 00 03
228- A5 E7 8D 01 03 A9 DD 20
230- 46 12 60 20 15 12 A9 11
238- 20 46 12 AD 00 03 85 E8
240- AD 01 03 85 E7 60 20 AA
248- 1E AD 0F 11 8D 7B 11 18
250- 20 22 1F 60 85 DF 66 E1
258- 90 02 66 E1 D0 B1 FF FF
```

```
                        0010 :*******************************
                        0020 :*                             *
                        0030 :*          SEQUE 1.0          *
                        0040 :*                             *
                        0050 :* MONOTONIC SEQUENCER PROGRAMS *
                        0060 :*                             *
                        0070 :*             BY              *
                        0080 :*    JOHN S. SIMONTON, JR.    *
                        0090 :*                             *
                        0100 :*(C) 1978 PAIA ELECTRONICS, INC*
                        0110 :*     ALL RIGHTS RESERVED     *
                        0120 :*                             *
                        0130 :*******************************
                        0140 :
                        0150 :DEFINE ADDRESSES OF LABELS
                        0160 :
                        0170 BEEP .DL 1F22
                        0180 DECD .DL 1F00
                        0190 CASS .DL 1EAA
                        0200 DBIT .DL 1E49
                        0210 SBIT .DL 1E25
                        0220 OUTP .DL 0840
                        0230 DSP  .DL 0820
                        0240 KBD  .DL 0810
                        0250 :
                        0260 MTB3 .DL 0303
                        0270 MTB2 .DL 0302
                        0280 MTB1 .DL 0301
                        0290 MTBL .DL 0300
                        0300 TTBL .DL 02C0
                        0310 :
                        0320 BUFF .DL 00F0
                        0330 KBUF .DL 00EC
                        0340 PBUF .DL 00EB
                        0350 MPNT .DL 00EA
                        0360 TPNT .DL 00E9
                        0370 MEND .DL 00E8
                        0380 TEND .DL 00E7
                        0390 TRNS .DL 00E6
                        0400 CNTR .DL 00E5
                        0410 TTRN .DL 00E4
                        0420 LSTL .DL 00E3
                        0430 STUS .DL 00E2
                        0440 TPO  .DL 00E1
                        0450 METF .DL 00E0
                        0460 MTRC .DL 00DF
                        0470 DUMY .DL 0003
                        0480 :
                        0490 :
                        0500 :
                        0510       OR 1000
                        0520 :
1000-  A9 00            0530 STAR LDA 00        :START / RESTART
1002-  85 E2            0540      STA *STUS     :CANCEL OPTIONS
1004-  A9 0C            0550      LDA 0C        :NRML COMMAND LINK
1006-  8D 7B 11         0560      STA ACTN+01   :PLACE COMMAND LINK
1009-  4C 18 11         0570      JMP COM       :JUMP TO COMMON
                        0580 :
                        0590 :NORMAL OPERATING MODE - DOES NOT ALTER
                        0600 :T-SEQUENCE OR M-SEQUENCE
                        0610 :
100C-  B0 05            0620 NRML BCS NRM1      :FIRST PASS THROUGH
100E-  85 E6            0630      STA *TRNS     :ZERO TRANSPOSE
1010-  8D 20 08         0640      STA DSP       :AND DISPLAYS
1013-  A5 EC            0650 NRM1 LDA *KBUF     :CHECK FOR NOTES
1015-  D0 04            0660 NRM2 BNE STOR      :ZERO- NO NEW KEY
1017-  A5 EB            0670      LDA *PBUF     :SO GET OLD KEY
1019-  29 3F            0680      AND 3F        :CLEAR BOTH FLAGS
101B-  85 EB            0690 STOR STA *PBUF     :SAVE AGAIN
101D-  60               0700      RTS           :AND RETURN
                        0710 :
                        0720 :PROGRAM TRANSPOSE MODE - NOTE PLAYED
                        0730 :IS "KILLED" WHEN KEY IS RELEASED
                        0740 :
101E-  B0 0A            0750 TLOD BCS TL1       :FIRST PASS, INITIALIZE
1020-  85 E6            0760      STA *TRNS     :ZERO TRANSPOSE FIGURE
1022-  85 EB            0770      STA *PBUF     :ZERO OUTPUT NOTE
1024-  85 E7            0780      STA *TEND     :ZERO TABLE END POINTER
1026-  A9 00            0790      LDA 00        :TURN T-SEQUE OPTION
1028-  85 E2            0800      STA *STUS     :ON
102A-  A6 E7            0810 TL1  LDX *TEND     :GET TRANSPOSE POINTER
102C-  8E 20 08         0820      STX DSP       :SHOW IT
102F-  A5 EC            0830      LDA *KBUF     :GET THE NOTE
1031-  F0 06            0840      BEQ TL2       :ZERO- NO KEY, SAVE
1033-  C5 EB            0850      CMP *PBUF     :KEY SAME AS LAST?
1035-  F0 05            0860      BEQ TRTN      :YES - LEAVE
1037-  E6 E7            0870      INC *TEND     :POINT TO NEXT LOCATION
1039-  9D C0 02         0880 TL2  STA TTBL,X    :SAVE TRANSPOSE
103C-  85 EB            0890 TRTN STA *PBUF     :AND OUTPUT AS NOTE
103E-  60               0900      RTS           :THEN RETURN
                        0910 :
                        0920 :PROGRAM SCORE MODE - USES REAL-TIME CLOCK
                        0930 :
103F-  20 84 11         0940 MSAV JSR MSV1      :CALL SAVE MODULE
1042-  E6 E5            0950      INC *CNTR     :INCREMENT THE TEMPO
1044-  60               0960      RTS           :COUNTER AND RETURN
                        0970 :
                        0980 :CONTINUE PLAY MODE - DOES NOT RESET
                        0990 :M-SEQUENCE OR T-SEQUENCE POINTERS
                        1000 :
1045-  38               1010 CNTU SEC           :SKIP INITIALIZATION
                        1020 :
                        1030 :REPEAT PLAY MODE - WHEN FIRST ENTERED
                        1040 :M-SEQ AND T-SEQ POINTERS ARE SET TO ZERO
                        1050 :BY THE PLAY MODULE (PLA1)
                        1060 :
1046-  20 AC 11         1070 RPLA JSR PLA1      :CALL PLAY MODULE
1049-  AD 14 11         1080      LDA STBL+14   :WAS THE PREVIOUS MODE
104C-  C5 E3            1090      CMP *LSTL     :MSAV (PROG. SCORE)?
104E-  D0 02            1100      BNE RPL1      :NO-SKIP INCREMENT
1050-  E6 E9            1110      INC *TPNT     :INC. T-SEQ POINTER
1052-  24 E2            1120 RPL1 BIT *STUS     :T-SEQ ASSERTED ?
1054-  30 0A            1130      BMI ROUT      :OPTION ON - LEAVE
1056-  A5 EC            1140      LDA *KBUF     :OPTION OFF- GET NOTE
1058-  F0 02            1150      BEQ OLDK      :AND IF NO NOTE, BRANCH
105A-  85 E4            1160      STA *TTRN     :SAVE NOTE FOR NEXT TIME
105C-  A5 E4            1170 OLDK LDA *TTRN     :GET LAST ACTIVE NOTE
105E-  85 E6            1180      STA *TRNS     :USE AS TRANSPOSE
1060-  E6 E5            1190 ROUT INC *CNTR     :INCREMENT TEMPO COUNTER
1062-  60               1200      RTS           :AND RETURN
                        1210 :
                        1220 :SINGLE PLAY MODE - WAITS FOR AGO KEY
                        1230 :THEN PLAYS SEQUENCE ONCE THROUGH
                        1240 :TRANSPOSED TO INDICATED KEY
                        1250 :
1063-  90 04            1260 SING BCC SNG1      :FIRST PASS, BRANCH
1065-  A5 EC            1270      LDA *KBUF     :AGO KEY DOWN ?
1067-  D0 DD            1280      BNE RPLA      :YES - PLAY SEQUENCE
1069-  20 46 10         1290 SNG1 JSR RPLA      :NO - "PLAY" THEN RETURN
106C-  A5 EA            1300      LDA *MPNT     :M-SEQ POINTER > 0 ?
106E-  D0 0B            1310      BNE SRTN      :YES - RETURN
1070-  A9 00            1320      LDA 00        :NO - PREPARE
1072-  85 E5            1330      STA *CNTR     :ZERO TEMPO COUNTER
1074-  A6 E8            1340      LDX *MEND     :POINT TO LAST NOTE
1076-  BD 01 03         1350      LDA MTB1,X    :OF M-SEQ AND GET IT
1079-  85 EB            1360      STA *PBUF     :PLACE IN PLAY BUFFER
107B-  60               1370 SRTN RTS           :THEN RETURN
                        1380 :
                        1390 :UP TEMPO AND DOWN TEMPO - COMMON PORTION
                        1400 :OF BOTH PROGRAMS ON PAGE 2
                        1410 :
107C-  A9 7E            1420 UTMP LDA 7E        :THE OP-CODE FOR ROR
107E-  D0 02            1430      BNE U/D       :BRANCH ALWAYS
1080-  A9 3E            1440 DTMP LDA 3E        :THE OP-CODE FOR ROL
1082-  4C 00 12         1450 U/D  JMP TCOM      :JUMP FOR THE REST
                        1460 :
                        1470 :REST MODE - EXTENDS NOTES OR UN-NOTES
                        1480 :WHEN IN PROGRAM EVENT MODE
                        1490 :
1085-  18               1500 REST CLC           :PREPARE FOR ADDITTION
1086-  A5 E5            1510      LDA *CNTR     :GET TEMPO COUNTER
1088-  65 E1            1520      ADC *TPO      :ADD TEMPO VALUE
108A-  85 E5            1530      STA *CNTR     :PUT COUNTER BACK
108C-  A5 E3            1540      LDA *LSTL     :AND RETURN TO
108E-  8D 7B 11         1550      STA ACTN+01   :PREVIOUS OPERATING
1091-  60               1560      RTS           :MODE
                        1570 :
                        1580 :STOP/STEP MODE - STOPS PLAY WITHOUT
                        1590 :CHANGING POINTERS. SINGLE STEPS THROUGH
                        1600 :SEQUENCE
                        1610 :
1092-  B0 0E            1620 STEP BCS STP1      :NOT FIRST PASS-BRANCH
1094-  A9 FF            1630      LDA 0FF       :SET TEMPO COUNTER AT
1096-  85 E5            1640      STA *CNTR     :"TIMED OUT" VALUE
```

```
1098-  20 B6 11   1650       JSR CONT     :CALL PART OF PLAY MODULE
109B-  8E 20 08   1660       STX DSP      :DISPLAY M-SEQ POINTER
109E-  A9 00      1670       LDA 00       :MAKE TRANSPOSE VALUE
10A0-  85 E6      1680       STA *TRNS    :EQUAL TO ZERO
10A2-  60         1690 STP1 RTS           :AND RETURN
                  1700 :
                  1710 :PROGRAM EVENT MODE - SAVES M-SEQUENCE
                  1720 :BUT SUBSTITUTES EVENT CLOCK FOR REAL-TIME
                  1730 :CLOCK
                  1740 :
10A3-  B0 02      1750 ESAV BCS ES1      :FIRST PASS, INITIALIZE
10A5-  85 E5      1760       STA *CNTR   :TEMPO COUNTER AS ZERO
10A7-  20 84 11   1770 ES1  JSR MSV1     :CALL SAVE MODULE
10AA-  A5 E5      1780       LDA *CNTR   :GET TEMPO COUNTER
10AC-  D0 05      1790       BNE EOUT    :NO ENTRY-RETURN
10AE-  18         1800       CLC         :PREPARE
10AF-  65 E1      1810       ADC *TPO    :ADD TEMPO VALUE
10B1-  85 E5      1820       STA *CNTR   :SAVE AS TEMPO COUNTER
10B3-  60         1830 EOUT RTS          :THEN RETURN
                  1840 :
                  1850 :OPTION MENU - RETURNS TO PREVIOUS
                  1860 :OPERATING MODE AFTER TURNING ON OR
                  1870 :CANCELLING OPTIONS
                  1880 :
10B4-  85 E9      1890 TBLM STA *TPNT    :T-SEQ POINTER TO BEG,
10B6-  A5 E2      1900       LDA *STUS   :ASSERT T-SEQ OPTION
10B8-  09 80      1910       ORA 80
10BA-  D0 0E      1920       BNE MCOM    :BRANCH ASLWAYS
10BC-  A5 E2      1930 MET  LDA *STUS    :TURN METRONOME ON
10BE-  09 40      1940       ORA 40
10C0-  D0 08      1950       BNE MCOM    :BRANCH ALWAYS
10C2-  A5 E2      1960 SYNC LDA *STUS    :TURN ON SYNC TO
10C4-  09 01      1970       ORA 01      :CLICK TRACK OPTION
10C6-  D0 02      1980       BNE MCOM    :BRANCH ALWAYS
10C8-  A9 00      1990 CNCL LDA 00       :PREPARE AND
10CA-  85 E2      2000 MCOM STA *STUS    :CANCEL ALL OPTIONS
10CC-  4C 0F 12   2010       JMP TCM1    :JUMP FOR THE REST
                  2020 :
                  2030 :CLICK MODE - SENDS CLICK TRACK TO TAPE
                  2040 :AGO KEYBOARD SCAN RATE IS TIMER
                  2050 :
10CF-  18         2060 CLIK CLC          :PREPARE TO SEND "0"
10D0-  20 25 1E   2070       JSR SBIT    :SEND IT
10D3-  60         2080       RTS         :RETURN FOR KEYBOARD DELAY
                  2090 :
                  2100 :METRONOME TEMPO CHANGE - PROGRAM ON PAGE 2
                  2110 :
10D4-  4C 54 12   2120 TCHG JMP TCH      :JUMP TO PROGRAM
                  2130 :
                  2140 :DUMP M&T-SEQ TO TAPE - PROGRAM ON PAGE 2
                  2150 :
10D7-  4C 20 12   2160 OTAP JMP TOUT     :JUMP TO PROGRAM
                  2170 :
                  2180 :LOAD M&T-SEQ FROM TAPE - PROGRAM ON PAGE 2
                  2190 :
10DA-  4C 33 12   2200 ITAP JMP TIN      :JUMP TO PROGRAM
                  2210 :
                  2220 :
                  2230 :COMMAND LINKS - LOW BYTE OF ADDRESS OF SUBS
                  2240 :
1100- 85 85 85 85 C2 BC B4 C8
1108- CF D4 00 7C DA D7 0C 0C
1110- 45 92 63 46 3F A3 1E 46
                  2250 :
                  2260       .OR 1118
                  2270 :
                  2280 :COMMON PROGRAM - DOES METRONOME WHEN ON
                  2290 :ADDS PLAY AND TRANSPOSE BUFFERS TO GET
                  2300 :OUTPUT NOTE, PLAYS NOTE, READS COMMAND
                  2310 :KEYBOARD AND JUMPS TO SELECTED MODE
                  2320 :SUBSTITUTES CLICK SYNCH FOR KEYBOARD
                  2330 :TIMING LOOP WHEN SYNC OPTION IS ASSERTED
                  2340 :
1118-  A5 E2      2350 COM  LDA *STUS    :CHECK OPTIONS
111A-  48         2360       PHA         :SAVE A COPY
111B-  0A         2370       ASL         :MERONOME ON ?
111C-  10 22      2380       BPL COM0    :NO - BRANCH
111E-  C6 DF      2390       DEC *MTRC   :DECREMENT METRONOME COUNTER
1120-  10 1E      2400       BPL COM0    :NOT <0 YET, BRANCH
1122-  A6 E1      2410       LDX *TPO    :TIME UP, GET TEMPO VALUE
1124-  CA         2420       DEX         :DECREMENT ONCE
1125-  86 DF      2430       STX *MTRC   :THEN SAVE AS COUNTER
1127-  A9 00      2440       LDA 00      :TO DETERMINE ALTERNATE DISPLAY
1129-  AA         2450       TAX         :CYCLE AND "PENDULUM" LEFT
112A-  18         2460       CLC         :PREPARE FOR ADDITTION
112B-  65 E0      2470       ADC *METF   :ADD FLIP-FLOP VALUE
112D-  85 E0      2480       STA *METF   :SAVE NEW VALUE
112F-  10 0C      2490       BPL MET1    :ALTERNATE? - DISPLAY
1131-  A5 EB      2500       LDA *PBUF   :OTHERWISE, GET OUTPUT
1133-  09 80      2510       ORA 80      :SET D7
1135-  85 EB      2520       STA *PBUF   :SAVE IN PLAY BUFFER
1137-  18         2530       CLC         :PREPARE AND
1138-  20 25 1E   2540       JSR SBIT    :CALL BEEP
113B-  A2 08      2550       LDX 08      :"PENDULUM" RIGHT
113D-  8E 20 08   2560 MET1 STX DSP      :SHOW PENDULUM
1140-  A5 E6      2570 COM0 LDA *TRNS    :IS THERE A TRANSPOSE ?
1142-  F0 03      2580       BEQ COM1    :NO - BRANCH
1144-  18         2590 TRAN CLC          :YES - PREPARE
1145-  69 A4      2600       ADC 0A4     :CALCULATE TRANSPOSE VALUE
1147-  18         2610 COM1 CLC          :MORE PREPARATION
1148-  65 EB      2620       ADC *PBUF   :CALCULATE NOTE
114A-  8D 40 08   2630 COUT STA OUTP     :PLAY NOTE
114D-  68         2640       PLA         :GET STUS (OPTION CODES)
114E-  6A         2650       ROR         :SYNC OPTION ON ?
114F-  90 06      2660       BCC KRED    :NO - SKIP
1151-  20 49 1E   2670       JSR DBIT    :WAIT FOR CLIK
1154-  4C 6D 11   2680       JMP CTRL    :SKIP READING AGO
1157-  2C 10 08   2690 KRED BIT KBD      :WAIT FOR DUMMY SCAN
115A-  10 FB      2700       BPL KRED    :LOOP UNTIL STARTED
115C-  AD 10 08   2710 KR2  LDA KBD      :WAIT FOR SCAN TO START
115F-  30 FB      2720       BMI KR2     :LOOP UNTIL STARTED
1161-  2C 10 08   2730 KR3  BIT KBD      :CHECK FOR KEYS DOWN
1164-  30 05      2740       BMI KRTN    :WHEN SCAN DONE, RETURN
1166-  50 F9      2750       BVC KR3     :CURRENT KEY NOT DOWN, LOOP
1168-  AD 10 08   2760       LDA KBD     :KEY DOWN, GET IT
116B-  85 EC      2770 KRTN STA *KBUF    :SAVE RESULT
116D-  20 00 1F   2780 CTRL JSR DECD     :GET COMMAND
1170-  B0 06      2790       BCS DO      :OLD COMMAND - DO IT
1172-  B9 00 11   2800       LDA STBL,Y  :NEW COMMAND - GET LINK
1175-  8D 7B 11   2810       STA ACTN+01 :PLACE LINK
1178-  A9 00      2820 DO   LDA 00       :THIS WILL BE HANDY
117A-  20 03 00   2830 ACTN JSR DUMY     :CALL OPERATING MODE
117D-  AD 7B 11   2840       LDA ACTN+01 :SAVE CURRENT COMMAND
1180-  85 E3      2850       STA *LSTL   :LINK FOR LATER
1182-  D0 94      2860       BNE COM     :AND LOOP ALWAYS
                  2870 :
                  2880 :SAVE MODULE - TAKES CARE OF ALTERNATELY
                  2890 :STACKING DURATIONS AND NOTES IN M-SEQUENCE
                  2900 :USES WHAT WILL BE "END OF SEQUENCE"
                  2910 :INDICATOR IN PLAY MODES AS POINTER
                  2920 :
1184-  B0 09      2930 MSV1 BCS MS1      :FIRST PASS?
1186-  8D 01 03   2940       STA MTBL+01 :YES-ZERO PROGRAM NOTE
1189-  85 E8      2950       STA *MEND   :ZERO M-SEQ POINTER
118B-  85 E6      2960       STA *TRNS   :ZERO TRANSPOSE
118D-  85 EB      2970       STA *PBUF   :ZERO OUTPUT NOTE
118F-  A5 E5      2980 MS1  LDA *CNTR    :GET TIME SINCE LAST NOTE
1191-  A6 E8      2990       LDX *MEND   :AND M-SEQ END POINTER
1193-  9D 00 03   3000       STA MTBL,X  :SAVE THE TIME
1196-  20 13 10   3010       JSR NRM1    :IN CASE NO KEYS DOWN
1199-  29 7F      3020       AND 7F      :CLEAR D7 IN OUTPUT NOTE
119B-  DD 01 03   3030       CMP MTB1,X  :SAME AS LAST NOTE?
119E-  F0 0B      3040       BEQ OUT     :YES, LEAVE
11A0-  E8         3050       INX         :NO, SAVE BY INCREMENTING
11A1-  E8         3060       INX         :M-SEQ POINTER TWICE
11A2-  86 E8      3070       STX *MEND   :AND SAVING AS END
11A4-  9D 01 03   3080       STA MTB1,X  :THEN SAVE NOTE
11A7-  A9 00      3090       LDA 00      :AND ZERO TIME SINCE
11A9-  85 E5      3100       STA *CNTR   :LAST NOTE
11AB-  60         3110 OUT  RTS          :AND RETURN
                  3120 :
                  3130 :PLAY MODULE - MANAGES M-SEQ AND T-SEQ
                  3140 :POINTERS AS WELL AS TEMPO CLOCK.
                  3150 :DETERMINES WHEN NOTES ARE TO BE PLAYED
                  3160 :
11AC-  B0 08      3170 PLA1 BCS CONT     :FIRST PASS ?
11AE-  85 E4      3180       STA *TTRN   :YES-ZERO TEMP. TRANSPOSE
11B0-  85 E9      3190 LP1  STA *TPNT    :ZERO T-SEQ POINTER
11B2-  85 EA      3200 LP2  STA *MPNT    :AND M-SEQ POINTER
11B4-  85 E5      3210       STA *CNTR   :AND CLOCK (TEMPO CONTER)
11B6-  A5 E5      3220 CONT LDA *CNTR    :GET CLOCK
11B8-  A4 E9      3230       LDY *TPNT   :GET T-SEQ POINTER
11BA-  A6 EA      3240       LDX *MPNT   :GET M-SEQ POINTER
11BC-  DD 02 03   3250       CMP MTB2,X  :TIME UP?
```

```
11BF-  90 15       3800       BCC PL1     :NO, BRANCH
11C1-  A9 00       3810       LDA 00      :YES, PREP. COUNTER, ETC.
11C3-  85 E5       3820       STA *CNTR   :FOR NEXT ACCUMULATION
11C5-  E8          3830       INX         :INCREMENT M-SEQ POINTER
11C6-  E8          3840       INX         :TWICE
11C7-  86 EA       3850       STX *MPNT   :AND SAVE NEW POINTER
11C9-  E4 E8       3860       CPX *MEND   :END OF M-SEQ?
11CB-  D0 09       3870       BNE PL1     :NO - BRANCH
11CD-  C8          3880       INY         :YES, INC T-SEQ POINTER
11CE-  C4 E7       3890       CPY *TEND   :END OF T-SEQ ?
11D0-  B0 DE       3900       BCS LP1     :YES-START T&M-SEQ AGAIN
11D2-  84 E9       3910       STY *TPNT   :NO-SAVE T-SEQ POINTER
11D4-  D0 DC       3920       BNE LP2     :BRANCH-START M-SEQ AGAIN
11D6-  BD 03 03    3930 PL1   LDA MTB3,X  :GET THE NOTE
11D9-  85 EB       3940       STA *PBUF   :SAVE IN PLAY BUFFER
11DB-  B9 C0 02    3950       LDA TTBL,Y  :GET TRANSPOSE
11DE-  85 E6       3960       STA *TRNS   :TO TRANSPOSE BUFFER
11E0-  60          3970       RTS         :RETURN
                   3980 :
                   3990 :TAPE TRANSFER PARAMETER TABLE
                   4000 :
                   4010 TAPE .HS FF00FF03C002C002
                   4020 :
                   4030       .OR 1200
                   4040 :
                   4050 :COMMON PORTION OF TEMPO UP & DOWN -
                   4060 :ROTATES RIGHT OR LEFT THE DURATIONS
                   4070 :SAVED WITH M-SEQUENCE
                   4080 :
1200-  8D 06 12    4090 TCOM STA PLAC     :PLACE ROR OR ROL OP CODE
1203-  A2 00       4100      LDX 00       :ZERO A COUNTER/POINTER
1205-  18          4110 TLP  CLC          :PREPARE
1206-  7E 02 03    4120 PLAC ROR MTB2,X   :ROTATE SAVED TEMPO
1209-  E8          4130      INX          :INCREMENT POINTER TWICE
120A-  E8          4140      INX          :TO POINT TO NEXT
120B-  E4 E8       4150      CPX *MEND    :END OF M-SEQ ?
120D-  D0 F6       4160      BNE TLP      :NO - LOOP FOR MORE
120F-  A5 E3       4170 TCM1 LDA *LSTL    :DONE, GET LINK AND
1211-  8D 7B 11    4180      STA ACTN+01  :SET UP FOR PREVIOUS MODE
1214-  60          4190      RTS          :THEN RETURN
                   4200 :
                   4210 :SET UP PROCEDURE FOR TAPE TRANSFER
                   4220 :
1215-  A2 07       4230 STTP LDX 07       :TRANSFER 7 BYTES
1217-  BD E1 11    4240 STP  LDA TAPE,X   :GET PARAMETER FROM TABLE
121A-  95 F0       4250      STA *BUFF,X  :PLACE IN POT-SHOT BUFFER
121C-  CA          4260      DEX          :POINT TO NEXT, MORE ?
121D-  D0 F8       4270      BNE STP      :YES - LOOP
121F-  60          4280      RTS          :NO - RETURN
                   4290 :
                   4300 :DUMP M-SEQ AND T-SEQ TO TAPE
                   4310 :
1220-  20 15 12    4320 TOUT JSR STTP     :SET UP FOR TRANSFER
1223-  A5 E8       4330      LDA *MEND    :SAVE M-SEQ END WITH
1225-  8D 00 03    4340      STA MTBL     :M&T-SEQUENCE
1228-  A5 E7       4350      LDA *TEND    :ALSO T-SEQUENCE END
122A-  8D 01 03    4360      STA MTB1
122D-  A9 DD       4370      LDA 0DD      :SET UP FOR DUMP
122F-  20 46 12    4380      JSR DOTP     :AND DO IT
1232-  60          4390      RTS          :THEN RETURN
                   4400 :
                   4410 :LOAD M-SEQ AND T-SEQ FROM TAPE
                   4420 :
1233-  20 15 12    4430 TIN  JSR STTP     :SET UP FOR TRANSFER
1236-  A9 11       4440      LDA 11       :SET UP FOR LOAD
1238-  20 46 12    4450      JSR DOTP     :AND DO IT
123B-  AD 00 03    4460      LDA MTBL     :PLACE M-SEQUENCE END
123E-  85 E8       4470      STA *MEND
1240-  AD 01 03    4480      LDA MTB1     :AND T-SEQUENCE END
1243-  85 E7       4490      STA *TEND
1245-  60          4500      RTS          :THEN RETURN
                   4510 :
                   4520 :PERFORM TAPE TRANSFER
                   4530 :
1246-  20 AA 1E    4540 DOTP JSR CASS     :CALL POT-SHOT
1249-  AD 0F 11    4550      LDA STBL+0F  :SET UP TO RETURN
124C-  8D 7B 11    4560      STA ACTN+01  :IN NORMAL MODE
124F-  18          4570      CLC          :PREPARE
1250-  20 22 1F    4580      JSR BEEP     :SIGNAL DONE
1253-  60          4590      RTS          :AND RETURN
                   4600 :
                   4610 :CHANGE METRONOME TEMPO
                   4620 :
1254-  85 DF       4630 TCH  STA *MTRC    :ZERO METRONOME CLOCK
1256-  66 E1       4640      ROR *TPO     :HALVE TEMPO VALUE
1258-  90 02       4650      BCC TCHR     :IF NOT ZERO, LEAVE
125A-  66 E1       4660      ROR *TPO     :ZERO, MAKE NOT ZERO
125C-  D0 B1       4670 TCHR BNE TCM1     :GO SET UP PREVIOUS MODE
                   4680 :
                   4690 END  .EN
                   4700
```

ECHO...ECHO....ECHO.....

A couple of issues ago, I said that we were going to look at a D/A that would allow those of you with exponential response synthesis equipment to begin playing with the computer software we have been discussing here. Then SEQUE ran longer than I thought it would, and we ran into logistics problems and .... In any case, it's not ready yet. Next time for sure.

Meantime, I've got some quickie code that I think you'll like, It's a program we call ECHO. I'll bet you think that ECHO echoes. It does.

It works in conjunction with an allocation algorithm (POLY from MUS 1 in this case, though something like Bob Yannes' SHAZAM could also be patched in to use this) and "follows" whatever data is being produced from QuASH channel #1, delaying it for a controllable period of time before playing it from a second channel, delaying again before playing on a third channel, and so on.

A convenient conceptual handle that may help you understand the "how-it-works" of ECHO might be a clock face. With only a second hand.

The numbers around the clock face represent memory locations and the second hand represents a pointer to these memory locations which, as it sweeps past each number, writes whatever note happens to be coming out of QuASH channel #1. This is really a funny clock, though, because in addition to the single second hand it has many minute hands that rotate at the same rate as the second hand. If the second hand is a "writing pointer", these funny minute hands are "reading pointers". Within some restrictions that we'll discuss shortly, we can have as many reading pointers as we like; the important feature is that each of these fast minute hands correspond to an additional QuASH channel.

Now as the clock runs, the writing pointer scans merrily through memory, writing the note that's in channel #1. In step behind it are the reading pointers, and as they point to successive memory locations they read them and place the result in the QuASH channel to which they correspond. Presto, echo.

In computerese, this kind of procedure is called a queue.

ECHO has a variety of software control features, and since I don't really know which of them are more important, we'll just plunge into the middle.

While ECHO always pulls the note that it's going to echo from channel #1, the first channel that the echo effect appears on doesn't have to be channel #2. Why? So that some channels can be set aside for polyphonic work while others are producing the echo.

Here's how. One piece of data that every polyphonic allocation subroutine must have is the number of output channels available for its use. POLY established the precedential name OUTS for this datum and set its location in a Paia 8700 as $EA.

Previously, we've always set this variable to represent the number of QuASH channels that were hardware supported. In a system which had a single QuASH, OUTS was set to contain $04 so that all available outputs were used for polyphonic allocation.

But OUTS may be set equal (may I please start saying "equal" instead of "contains"? It's not strictly true, but much less cumbersome.) to a number less than the number of hardware supported channels and the result will be to reserve some channels. In a system with two QuASH (for example) OUTS could be set equal to $05 and the result would be that the upper 3 channels (6 - 8) will not have keyboard activations directly assigned to them. POLY (or whatever) doesn't know they're there.

So we can use them for other things. Like echo channels.

ECHO, in its turn, must know how many channels it has to work with. The location labeled ECCO ($BB) serves this function, and in most cases will be set equal to the number of remaining channels.

To give a final example; if we make OUTS equal to $03 and ECCO equal to $05, we've produced a system which has 3 polyphonic channels (the first three) with channels 4 through 8 echoing, in sequence, the notes that appear on polyphonic channel #1.

I would be less than candid if I didn't forewarn you that successful use of a system which combines both polyphonic and echo channels requires a thorough understanding of the allocation algorithm being used as well as a certain manual and mental dexterity. It's best to start playing with a configuration which has only one channel available to POLY and the remainder used as echo channels. With practice, you can progress from there.

DELAY CONTROLS

As you certainly know by now, all timing in our system references back to the scan rate of the keyboard, and ECHO has associated with it a variable

labeled EDLY ($BC) which regulates how fast (in terms of keyboard scans) the hands in our clock analogy (the reading and writing pointers) advance from one memory location to the next, which in turn contributes to how long the echo delay is.

If we set EDLY equal to $01, the echoing routine is invoked after every keyboard scan (which is variable, but typically will be every 10 to 50 milliseconds). Making EDLY equal to $02 means that the routine is used on alternate scans which, if everything else is equal, will produce an echo delay twice as long.

Notice that this affects only the ECHO and does nothing to alter POLY's allocating channels after every keyboard scan. This is important because when changing the value of EDLY you should be aware that if you skip more than about 8 scans before invoking ECHO, it may miss some keyboard activity in a fast riff. The notes will still play through the polyphonic channels, but won't be echoed.

A second variable also interacts with EDLY to detirmine the echo delay. OFST ($BD) controls the offset between the pointers into the echo queue. Going back to the clock metaphore, it detirmines how "far apart" the hands on the clock are. The farther apart they are (the bigger the number in OFST), the greater will be the echo delay.

Like EDLY, there are some caveats that go with OFST. The echo buffer (queue) area of memory is 64 bytes on page 1. You don't want to come up with too many pointers (controlled by ECCO, remember) that are too far apart or they will represent a memory area larger than that set aside. The result of that is far from disastrous, but it will cause things like the high order channels echoing much sooner than you expected, as the reading pointers for those channels "wrap around" past the writing pointer. But, as we've decided here in the past, the difference between noise and a neat effect is often nothing more than a creative mind.

Control of the time delay involved in the echo is important for reasons that you might not first think about, because like any device (or now software) that messes with the subjective flow of time, echo offers a variety of totally different effects depending on how long a time we are talking about.

For example, if the delay is very short, as when both EDLY and OFST are set to $01, the effect will not even be percieved as an echo, but rather as a "thickening" of the voice (voice doubling, actually). It's a lot like phasing or flanging, except that with those techniques the predominant effect is frequently that the subjective flow of time is cyclicly changing.

Longer delays (EDLY = $01 and OFST = $08) produce the types of effects which give ECHO its name. Echoplex type echoing. There is a major difference, though, in that with conventional echo devices you can only echo in a voice that is essentially the same as the starting voice. Here, the echoes can be anything, and there's no way to appreciate the power that this implies without working with it.

When the delays get very long (EDLY = $02 and OFST = $10) you find yourself playing with an instrument that allows you to play rounds with yourself. Also, of course, in different voices.

Because the character of the instrument is so greatly influenced by delay times, and because the different characters can so frequently be used in the same musical performance, we've added a means of quickly switching from one set of operating parameters to another. Four of these presets are provided by pads 0-3 on the command keyboard. Touching one of these pads causes ECHO to get the requested set of parameters from a table that lives in memory $9A - $A9 and place them in the locations referenced by the rest of the program. The pre-sets that are in place in the listing which follows are:

COMMAND KEY	POLY CHANS	ECHO CHANS	TIME DELAY (KBD SCNS)
0	1	7	1
1	1	7	8
2	1	3	16
3	1	3	32

Notice a couple of things here. First, if you're using a system with only a single QuASH (a P4700/J or its equivalent) it doesn't matter that there are more echo channels than there are hardware channels; the last four iterations simply won't have the hardware to voice them. Secondly, observe that when we got to longer delays we cut back on the number of echo channels so as to circumvent the "too many channels too far apart" problem that we looked at earlier.

You can substitute your own presets for those shown simply by altering or replacing the values shown. Here is a map of locations that will make that a little easier:

	PRESET #			
	0	1	2	3
OUTS	$9A	$9E	$A2	$A6
ECCO	$9B	$9F	$A3	$A7
EDLY	$9C	$A0	$A4	$A8
OFST	$9D	$A1	$A5	$A9

With some experimentation you will find echo presets which seem to complement each other particularly well. You will inevitably get to where you use a specific set of presets for each particular song, not only changing presets throughout the song but within a riff or phrase. This can create some neat effects such as having an initially long delay set and, in the middle of the echo chain, hit a faster preset to initiate a burst of echoes. Or, have one preset for the "voice doubling" characteristics we discussed. Then you can switch between echoes for special effects and doubling for use on bass lines or solos.

Actually, there is a lot of power hidden in this program that can be liberated with innovative patching, voicing, and mixing. How about having a chain of voices which are all related but slightly different, such as having higher Q on the filters as the echo is passed on. Or changing envelope times so the first echoes have sharp attacks and delays and later voices have increasingly softer envelopes. Here's a good one- progressively detune each voice so you get a spiraling echo, or the echoes sequence upscale (or downscale). Completely different voices can be used, and this technique really works well on the long delays for doing rounds.

Just playing with the mixing or panning of the normal echo voices can entertain you for hours. Have the echoes pan across the stereo field, or bounce back and forth. Or have the echoes begin to fade out, but set the last or next to last voice at a

higher level.

You can also use a multi- voice setup with only a few of the outputs driving voices. Set up the computer to provide (for example) one poly voice and seven echo voices, but only use channels 1, 4, 5, and 8 to drive oscillators. Work with various combinations here; each is a completely different rhythm and could easily provide a rhythmic basis for a whole piece.

Well, by now you are probably ready to dig into the program, so here is the listing.

LOADING THE PROGRAM

As with other programs that we've examined in the past, ECHO may be hand-loaded using the 8700 computer's monitor, but first set the monitor stack pointer:

O-E-D-DISP-F-F-ENT

and the user's stack pointer and status register:

O-F-E-DISP-F-F-ENT-0-0-ENT

and then load the program:

O-0-0-DISP-2-0-ENT-2-1-ENT-8-D-ENT- (etc.)

and don't forget this data base information:

```
088-  20  21  0D  4C  C0  FF  C9  07
090-  D0  05  A0  5C  20  52  0D  4C
098-  10  10  01  07  01  01  01  07
0A0-  01  08  01  03  02  08  01  03
0A8-  02  10
0B8-  FF  FF  01  03  02  04
0E8-  40  20  01
```

After loading (and before running) the program and data should be dumped to tape (from location $000 to $0EC) using this sequence:

O-0-0-0-0-0-E-C-0-1-D-D-TAPE

When this tape is loaded in the future, it should be loaded from $000 to $0EC so that the presets will be loaded along with the program.

```
                0010  :*********************************
                0020  :*                               *
                0030  :*         ECHO 0.31             *
                0040  :*                               *
                0050  :*   POLYPHONIC VOICE QUEUING     *
                0060  :*                               *
                0070  :*            BY                 *
                0080  :*       'JOHN SIMONTON          *
                0090  :*                               *
                0100  :*(C) 1979 PAIA ELECTRONICS, INC*
                0110  :*     ALL RIGHTS RESERVED       *
                0120  :*                               *
                0130  :*********************************
                0140  :
                0490  :
                0500  :INITIALIZE SYSTEM, CLEAR OUTPUT BUFFERS AND ECHO BUFFER
                0510  :
000-  20 21 0D  0520  STAR JSR INIT    :CALL MUS1 INITIALIZATION
003-  A2 FF     0530       LDX 0FF     :PREPARE TO SET STACK POINTER
005-  9A        0540       TXS         :SET STACK TO TOP OF PAGE
006-  A9 00     0550  EBZR LDA 00      :PREPARE TO ZERO OUT ECHO BUFFER
008-  A2 3F     0560       LDX 3F      :POINTER TO END OF ECHO BUFFER
00A-  9D 00 02  0570  ILP  STA EBUF,X  :ZERO ECHO BUFFER LOCATION
00D-  CA        0580       DEX         :POINT TO NEXT LOCATION
00E-  10 FA     0590       BPL ILP     :NOT DONE YET, LOOP
010-  20 71 0D  0600  ECHO JSR POLY    :CALL MUS1 POLYPHONIC ALLOCATION
                0630  :
                0640  :DETERMINE ADDRESS OF THE FIRST CHANNEL AVAILABLE
                0650  :FOR ECHO USE
                0660  :
013-  A0 0F     0670       LDY 0F      :OFFSET TO FIRST OUT-BUF LOCATION
015-  A6 EA     0680       LDX *OUTS   :NUMBER OF POLYPHONIC CHANNELS
017-  88        0690  LP0  DEY         :POINT TO NEXT OUTPUT CHANNEL
018-  CA        0700       DEX         :ONE LESS POLY CHANNEL
019-  D0 FC     0710       BNE LP0     :ALL POLY CHANS NOT USED, LOOP
01B-  84 EB     0720       STY *OUTT   :SAVE FIRST ECHO POINTER FOR LATER
                0730  :
                0740  :ADVANCE ECHO BUFFER POINTER AND ADJUST IF NECESSARY
                0750  :
01D-  A6 BE     0760       LDX *EPNT   :GET CURRENT ECHO BUFFER POINTER
01F-  C6 EC     0770       DEC *CNTR   :DECREMENT TIMER
021-  D0 09     0780       BNE GETN    :TIME NOT UP, BRANCH
023-  A5 BC     0790       LDA *EDLY   :TIME UP, RE-INIT TIMER VALUE
025-  85 EC     0800       STA *CNTR   :RE-INITIALIZE TIMER
027-  CA        0810       DEX         :POINT TO NEXT
028-  10 02     0820       BPL GETN    :BRANCH IF STILL WITHIN BUFFER AREA
02A-  A2 3F     0830       LDX 3F      :OTHERWISE, RE-INIT POINTER
02C-  86 BE     0840  GETN STX *EPNT   :SAVE NEW POINTER
                0850  :
                0860  :PUT CURRENT CHANNEL 1 NOTE IN ECHO BUFFER AND
                0870  :PREPARE ECHO CHANNEL COUNTER
                0880  :
02E-  A5 DF     0890       LDA *CHN1   :GET CHANNEL 1 NOTE
030-  9D 00 02  0900       STA EBUF,X  :SAVE IN ECHO BUFFER
033-  A5 BB     0910       LDA *ECCO   :GET NUMBER OF ECHO CHANNELS
035-  85 BA     0920       STA *TEMP   :SAVE AS COUNTER
                0930  :
                0940  :CALCULATE SUCCESSIVE ECHO BUFFER LOCATIONS AND
                0950  :ADJUST AS NECESSARY
                0960  :
037-  8A        0970  LP1  TXA         :ECHO BUFFER POINTER TO ACCUMULATOR
038-  18        0980       CLC         :PREPARE FOR ADDITTION
039-  65 BD     0990       ADC *OFST   :CALCULATE NEXT LOCATION
03B-  C9 40     1000       CMP 40      :STILL WITHIN ECHO BUFFER?
03D-  90 03     1010       BCC SAVE    :YES, BRANCH TO CONTINUE
03F-  38        1020       SEC         :NO, SET CARRY FOR SUBTRACTION
040-  E9 40     1030       SBC 40      :AND ADJUST POINTER
042-  AA        1040  SAVE TAX         :PUT POINTER IN PLACE
                1050  :
                1060  :THEN PULL NOTES FROM ROTATED ECHO BUFFER LOCATIONS
                1070  :AND PLACE IN ECHO CHANNELS OF OUTPUT BUFFER (NTBL)
                1080  :
043-  BD 00 02  1090       LDA EBUF,X  :GET NOTE FROM ECHO BUFFER
046-  99 D0 00  1100       STA NTBL,Y  :PLACE TO OUTPUT CHANNEL
049-  88        1110       DEY         :POINT TO NEXT OUTPUT CHANNEL
04A-  C6 BA     1120       DEC *TEMP   :ONE LESS ECHO CHANNEL
04C-  D0 E9     1130       BNE LP1     :BUT SOME LEFT, LOOP
                1140  :
                1150  :NOTES ARE PLAYED BY CALLING THE QUASH DRIVER (NOTE).
                1160  :FINALLY, ECHO OUTPUT CHANNELS ARE CLEARED SO AS NOT
                1170  :TO CONFUSE POLY WHEN CALLED
                1180  :
04E-  20 2B 0D  1190       JSR NOTE    :CALL MUS1 QUASH DRIVERS, ETC.
051-  A4 EB     1200       LDY *OUTT   :GET FIRST ECHO CHANNEL POINTER
053-  A6 BB     1210       LDX *ECCO   :GET # OF ECHO CHANNELS
055-  A9 00     1220       LDA 00      :PREPARE TO ZERO
057-  99 D0 00  1230  LP2  STA NTBL,Y  :ZERO ECHO OUTPUT CHANNEL
05A-  88        1240       DEY         :POINT TO NEXT OUTPUT
05B-  CA        1250       DEX         :ONE LESS ECHO CHANNEL
05C-  D0 F9     1260       BNE LP2     :SOME LEFT, LOOP
                1270  :
                1280  :READ COMMANDS. 0-3; PRESETS, 4-INITIALIZE SYSTEM
                1290  :5-CLEAR ECHO, 6-BREAK, 7-TUNE
                1300  :
05E-  20 00 0F  1310       JSR DECD    :READ COMMAND KEYBOARD
061-  C9 04     1320       CMP 04      :IS COMMAND A PRE-SET?
063-  10 1B     1330       BPL NEXT    :NO, BRANCH FOR NEXT TEST
                1340  :
                1350  :THE COMMAND IS TO CALL UP A PRE-SET.  AFTER CALUCLATING
                1360  :THE BASE ADDRESS OF THE PRE-SETS CALLED FOR, THE PRESET
                1370  :VALUES ARE TRANSFERED TO THEIR RESPECTIVE LOCATIONS
                1380  :AS ACTIVE PARAMETERS. NOTE THAT THE NUMBER OF
                1390  :CHANNELS ALLOCATED TO POLY USAGE (OUTS - $00EA) IS IN
                1400  :NON-CONTIGUOUS LOCATION AND MUST BE HANDLED SEPARATELY
                1410  :NOTE THAT THE CONTIGUOUS LOCATION *TEMP IS USED AS A
                1420  :DUMMY VARIABLE AT THIS POINT
                1430  :
065-  8C 20 08  1440       STY DISP    :SHOW PRESET
068-  A9 FF     1450       LDA 0FF     :ONE LESS THAN PRESETS BASE ADDRESS
06A-  18        1460  LP3  CLC         :PREPARE FOR CALCULATION
06B-  69 04     1470       ADC 04      :THERE ARE 4 PRESET VARIABLES
06D-  88        1480       DEY         :POINT TO NEXT PRESET BASE
06E-  10 FA     1490       BPL LP3     :IF NOT THIS PRESET, LOOP
070-  AA        1500       TAX         :PUT POINTER CALCULATED TO X
071-  A0 03     1510       LDY 03      :4 PRESETS, WILL COUNT TO -1
073-  B5 9A     1520  LP4  LDA *PRST,X :GET PRE-SET DATA
075-  99 BA 00  1530       STA TEMP,Y  :AND PLACE AS ACTIVE PARAMETER
078-  CA        1540       DEX         :POINT TO NEXT PRESET DATA
079-  88        1550       DEY         :AND NEXT ACTIVE PARAMETER
07A-  10 F7     1560       BPL LP4     :IF NOT YET DONE, LOOP
07C-  85 EA     1570       STA *OUTS   :SAVE THE MAVERICK PARAMETER
07E-  30 90     1580       BMI ECHO    :BRANCH ALWAYS
                1590  :
080-  F0 7E     1600  NEXT BEQ STAR    :COMMAND IS FOR CLEAR, BRANCH
082-  C9 06     1610       CMP 06      :IS COMMAND 5 (CLEAR ECHO) OR 6 (BRK)?
084-  30 80     1620       BMI EBZR    :COMMAND IS CLEAR ECHO, BRANCH
086-  D0 06     1630       BNE NXT0    :COMMAND IS NOT BRK, BRANCH
088-  20 21 0D  1640       JSR INIT    :SHUT DOWN SYNTHESIZER
08B-  4C C0 FF  1650       JMP BRAK    :AND RETURN TO MONITOR
08E-  C9 07     1660  NXT0 CMP 07      :IS COMMAND TUNE?
090-  D0 05     1670       BNE BRDG    :A BRANCH TOO FAR
092-  A0 5C     1680       LDY 5C      :PREPARE TO TUNE TO MIDDLE C
094-  20 52 0D  1690       JSR FILL    :SEE MUS 1.0 DOCUMENTATION
097-  4C 10 10  1700  BRDG JMP ECHO    :PLAY ON AND ON AND ON
                1710  :
                1720  :SET-UP VARIABLES FOR MUS1
                1730       .OR 10BA    :INITIAL PRE-SET
                1740       .HS 01030204
                1750       .OR 10E8    :SYSTEM CONTROL AND QUASH DELAY
                1760       .HS 402001  :AND OUTS
                1770  :AND PRESETS
                1780       .OR 109A
                1790       .HS 01070101
                1800       .HS 01070108
                1810       .HS 01030208
                1820       .HS 01030210
                1830  :
                1840  END  .EN
```

NOTES

CONTROLLING EXPONENTIAL SYSTEMS

The two most common questions I hear about the computer - based synthesizer systems we've been developing here are:

1) How do I use it with my exponential synthesis gear?

and

2) How do I use it with my Razmataz RMT-80 computer?

The answer to the second question is going to have to wait just a bit longer (though I expect to have a surprising answer soon).

The answer to the first question is what we're going to focus on this time by looking at a Digital to Analog converter that is designed to be compatible with almost every synthesizer in the world with the exception of the linear holdouts- Paia, Yamaha CS series, Unicord, some EML; you know who they are. For them, you use the stuff we've already covered.

By way of a very short review, the differences between D/As that are to be used with linear response elements and those that are to work with Moog, Arp, or any other exponential system are not great from a basic conceptual standpoint. A binary number is fed in one end, and a DC control voltage comes out the other. But, they do differ greatly in the character of the voltage that comes out.

For linear response equipment, the D/A must produce an output that has an exponential character- as the control voltage increases, the incremental change in voltage must also increase.

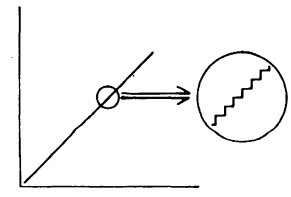
Since exponential response equipment has analog circuitry built into the front end of each control input which "bends" the linear control signal into an exponential curve, a D/A that is to be used with this equipment must produce a linear output voltage function. That is, the incremental change in output voltage must be constant. See figure 1.

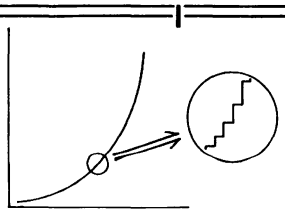
One of the nicer things about this linear D/A is that it's common, the kind that most applications require. Since it is common, we have a large number of parts to choose from. From that large number we've selected a "5008" type which is made by a number of manufacturers. When Signetics makes it and houses it in a 16 pin plastic package it becomes an NE5008N.

Inside, this chip is relatively simple. It looks like figure 2. The transistors shown are each a current source and the values of the resistors in the matrix that their emitters are tied to are such that if the source associated with D0 is pumping some current (i), the one that corresponds to D1 will pump twice that (2i). Similarly, the source that goes along with data bit D2 produces twice what the previous one did (4i), and so on.

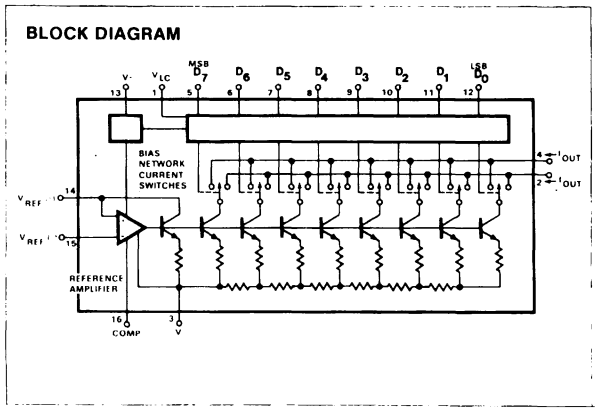
In response to a bit being set, the current produced by the source associated with that bit is switched so that instead of appearing at pin 2 of the IC it appears at pin 4 (Iout). At any given instant, this output current will be the sum of the currents corresponding to each input bit which has been set.

To turn this chip into a "system" that accepts data at the input and controls a synthesizer at the output, we need to add such niceties as latches to hold the data that the computer sent out, an I/V (current to voltage) converter to change the 5008's current output to a voltage that our synthesizers will like, and





2



other bells and whistles as available.

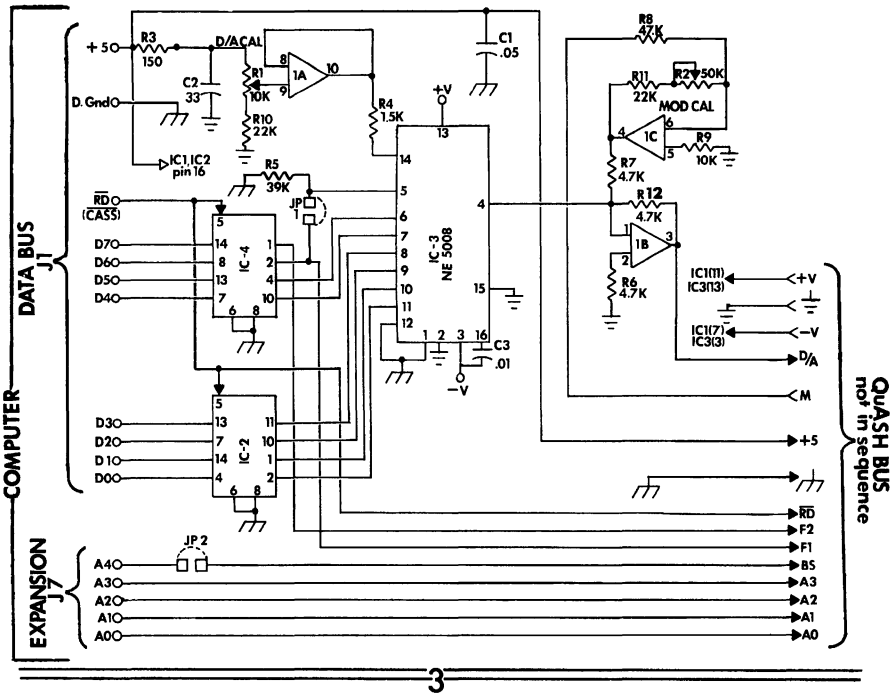
When we do all this, the design looks like figure 3. It's pretty straight - forward. We've used 4042's to latch the data coming in and the $\overline{RD}$ line is the strobe on these latches which, when low, allows the data present at their inputs to appear at the outputs. When $\overline{RD}$ is high, whatever data was present at the latch inputs when the line went high will be held at the outputs. Notice that the two most significant data bits follow our previous protocols in that they come out simply as flags rather than being presented to the converter circuitry. But notice also the jumper JP1 which, as we'll see later, can be used to double the range of the D/A (although at what might be an unacceptably high cost).

We've used a 4136 quad op-amp to provide all of the analog support that the 5008 needs; one stage serves as a buffer between the calibration trimmer and the 5008's Vref input (IC1a), another comprises a current to voltage converter (IC1b), and a third is an inverting summing amplifier that allows a modulation input (IC1c).

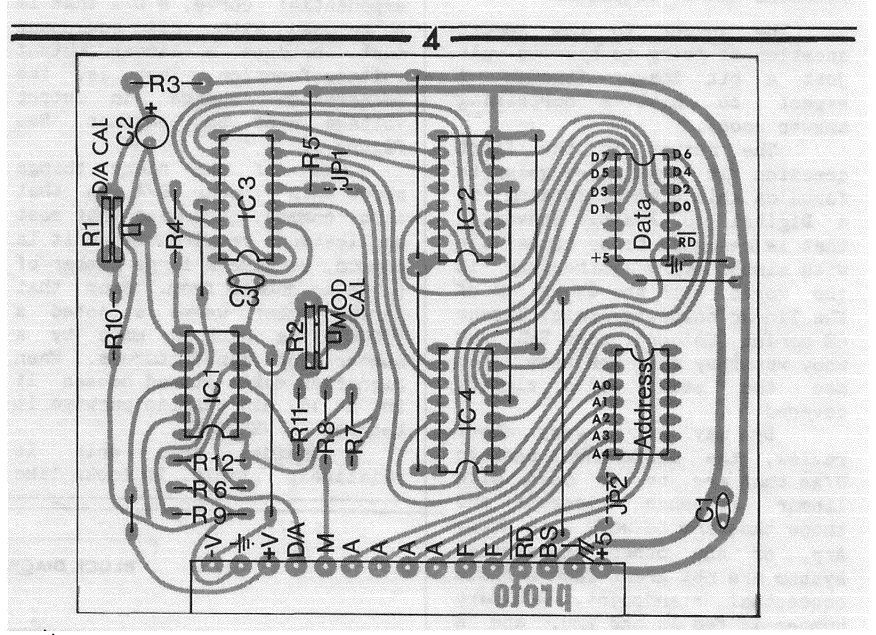
With the exception of the standard "be tidy" caveats, there's nothing very critical about this D/A system and you can build it using whatever construction techniques appeal to you, but the board which is available from Paia has enough interesting features that it's worth taking a special look at it. Check out figure 4.

I suppose the most interesting thing is the way the input, output, and control lines are configured. Notice that the connections to the computer all appear on two 14 pin dip outlines (J1 and J2), while connections to the synthesizer (including some computer address lines that QuASH in an expanded system will need; see "In Pursuit Of The Wild QuASH", Polyphony July '77, page 19) come out to the 15 pin Molex-type edge connector (J3).

We've already examined in general terms how this type of D/A connects at the computer side (see "The Polyphonic Synthesizer", Polyphony February '78, page 28). If the computer you're using is a Paia 8700 (which is not a bad idea since it has some useful music software to support



3



4

it), these connections couldn't be simpler - there is a one to one correspondence between J1 and J2 and the connectors they mate with on the computer. Standard pre-terminated jumpers are used to connect the two. No soldering.

The wiring to the "synthesizer" side is also arranged to acknowledge the fact that almost everyone will want to expand to a multi-channel system sooner or later (it's actually what the computer stuff is best at!), so the Molex wiring is the same as that found on QuASH modules.

All of this means that from an inter-wiring standpoint, a fully expanded system is exceptionally easy to implement. Figure 5 shows you how.

Calibration of the 8785 D/A consists of adjusting the D/A CAL trimmer (R1) so that octave changes in the input data produce

octave changes in the module being controlled; this can easily be done by ear. The MOD CAL trimmer (R2) should be set so that a one volt (or whatever represents one octave in your system) change at the modulation input produces a one octave change in the controlled element.

Before we wrap this column up, there are some little detail things that really need to be mentioned.

Going back to the schematic for a minute, observe that there are two "programming" jumpers (JP1 and JP2) indicated on the circuit board.

As we've mentioned again and again, the Paia protocols use the least significant 6 bits of an 8 bit word to specify an analog parameter while the two most significant bits are flags (D6 is used as a gate, and D7 is a general purpose control bit which QuASH recognize as a portamento control bit). Since the 5008 is an eight bit converter, obviously some bits will not be used. I decided to permanently not use the least significant bit (LSB) of the converter (pin 12) by grounding it. The only effect of this is to slide all the lines of the controller "up one" as far as the 5008 is concerned, and it has no electrical effect that we need to worry about.

The other unused 5008 bit is then it's MSB (most significant bit - D7, pin 5) and if the jumper JP1 is not in place, this bit is in fact not used. But, if you are one of those people for whom nothing is ever enough, you have the option of installing the jumper. This means that the MSB of the 5008 is tied to data bit D6, effectively doubling the range of the D/A from 64 notes (over 5 octaves) to 128 notes (almost 11 octaves).

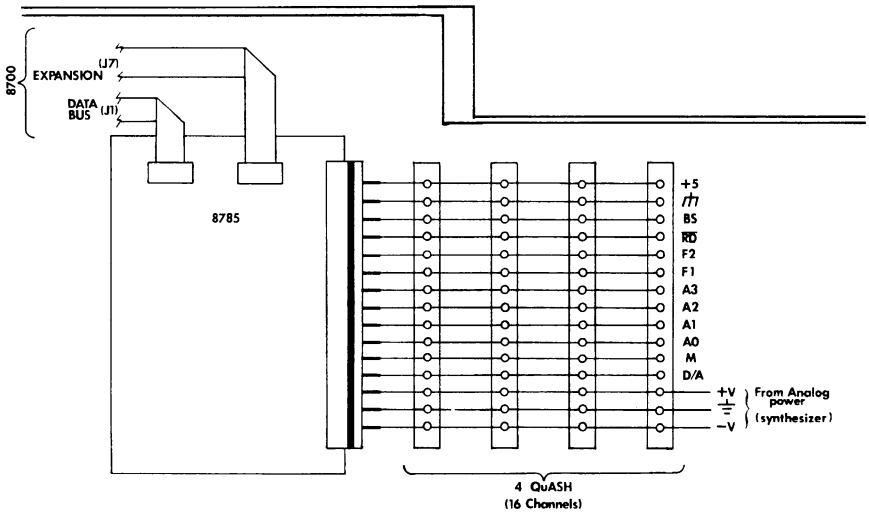
The cost of this "simple" modification is much greater than just a piece of wire, though, because if the option is selected the system is no longer compatible with our existing software (which might be just fine for your purposes). Maybe worse than that, it's no longer compatible with QuASHes either. But if you need it, it's there.

A second jumper (JP2) is meant to be used in systems with 4 or more QuASH and causes the fifth address bit from the computer to serve as the Bank Select (BS) line (see "In Pursuit Of The Wild QuASH" referenced earlier).

Something else to worry about is grounding. At some point in the system, digital power ground (recognized as a chassis ground symbol) and analog power ground (recognized as an earth ground symbol) must be tied together. However, they must have a common connection at only one point. Otherwise you run the risk of ground loop problems. I recommend that these two grounds be tied together at the Molex connector of the D/A, as shown in figure 5.

Finally, Moog "S" triggers must be pulled to ground rather than accepting the high logic level that our trigger outputs provide. The simple circuit in figure 6 takes care of this using almost any NPN transistor you happen to have laying around.

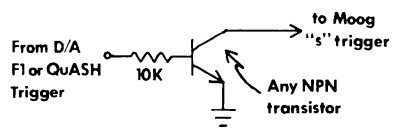
Synthesizers that have both "gate" and "trigger" inputs can use the scheme shown in figure 7 to derive both of these signals from the single gate that our D/A produces.



5

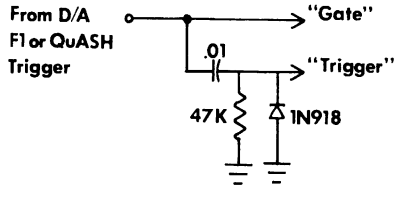
A complete kit for the Linear D/A including printed circuit board, sockets, headers and edge connector is available from Paia Electronics, Box 14359, Oklahoma City, OK 73113. Order #8785 Linear D/A. $22.95 ppd.

6



MOOG "S" TRIGGER ADAPTER.

7



"GATE" & "TRIGGER" ADAPTER.

NOTES

DIGITIZERS

There are plenty of times when a switch is a great way to control things- like when you want to turn something on and off, or select a preset. But when you're just playing around looking for the right sound, there's nothing quite like a knob. Unless it's a joystick.

Knob or joystick, either one- we need some way to digitize it's position so a computer can read, save and manipulate the data various ways. And preferably it should be a cheap and simple way.

We need something we'll call a digitizer. It's an analog-to-digital converter, really; the only reason I don't think we should call it an ADC is that we reserve that term for something more elaborate than what we are getting into. This is really simple.

In every electronic scheme that I know of to convert an analog parameter to a digital one there is a thing called a comparator. See figure 1. The thing it compares are the voltages at its "+" and "-" inputs. If the voltage at the "+" input is greater, the output is at a high voltage. If the "-" input is greater, the output is driven to a low voltage.

The elaborate ADC's use the comparator as only a small part of a larger circuit that will probably look something like figure 2. When it's time to quantize the voltage to be measured, the counter is reset and its digital output goes to zero. Because of this, the D/A puts out a low voltage (in this scheme you must first have a digital to analog conversion before you can have the reverse). The output of the D/A will probably be lower than the voltage that is being measured, so the output of the comparator is high and allows pulses to pass from the clock through the NAND gate to the counter. The counter counts up and, as it does, the output of the D/A increases. When the output of the D/A exceeds the voltage to be measured, the comparators output goes low and clock pulses can no longer pass through the gate to the counter. At that point, the counter's output is a digital representation of the analog voltage being measured.

There are a number of variations on this design that have to do with the way the counter works, and in a computer based system it is common to replace both the clock and counter with software. Unfortunately, the common features of all these variations are modest complexity and/or relatively slow conversion rate.

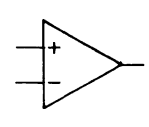
hardware

Now, for a <u>really simple</u> digitizer, take a look at figure 3. Since the resistors in the divider that detirmines the reference voltage (Vr) are equal, the digital output is a 1 (high) if the voltage is greater than 1/2 the supply voltage and 0 if the input is less then Vs/2. I know what you're thinking, and you're right. A one bit digitizer isn't exactly an improvement over a switch in most cases.

OK, let's add another stage. Only on this one, let's make the reference voltage a function of the output state of the first stage. Schematically, this is represented in figure 4.

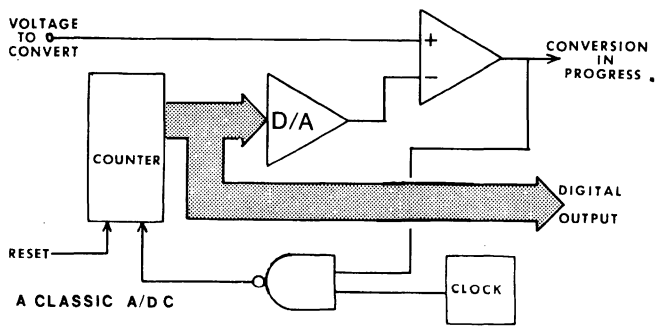
In order to easily see how this circuit works, you have to assume that Vr1 (the voltage at the junction of the two R1's) is constant at Vs/2. In fact, this voltage will change as the comparator output D1 changes

1



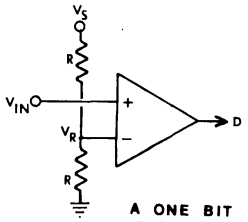
A COMPARATOR

2



A CLASSIC A/D C

3



A ONE BIT DIGITIZER

and alternately sinks or sources current through the two resistors, R2. But as long as the value R1 is kept much lower than the value R2 (the lower the better, at least 1/10), the change in Vr1 will not be too significant.

Imagine that a voltage which is increasing from ground to supply is applied to the input of the digitizer. When at ground, the voltage is less than Vs/2, so D1 is low (ground). The two R2's now form a voltage divider at the junction of which is a voltage equal to 1/2 of Vs/2, or 1/4 of the supply voltage (Vs/4). This voltage (Vs/4) is the reference voltage for the new stage. Since we said that our input voltage was initially at ground (which is less than Vs/4), the output of the new stage is also low. In binary, the output of the two stages is 00. An equivalent circuit would look like figure 5.

Now we increase the input voltage and, as it exceeds Vs/4, the output of the new stage changes from low to high. That's all that happens; the binary output of the two stages is now 01.

We continue to increase the input voltage and, as it exceeds Vs/2, the output of the first stage goes high. But, that's not all, because with the output high (at Vs), an equivalent circuit of the voltage divider that forms the reference for the new stage looks like figure 6. Since the input voltage is less than 3/4 of the supply voltage, the new stage changes state back to low and all is once again stable with a binary output of 10.

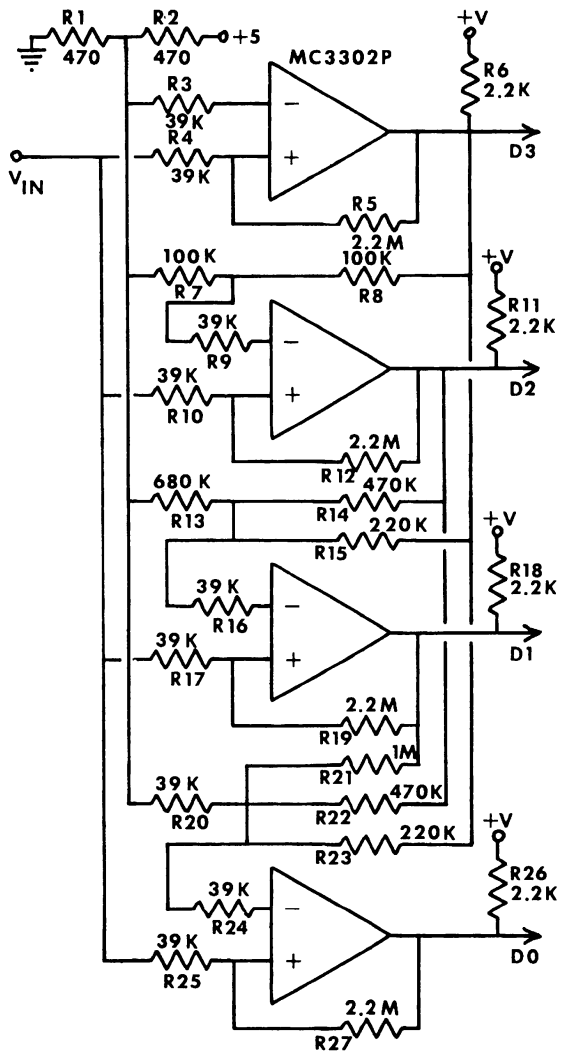
Increasing the input voltage further will exceed 3Vs/4. The new stage again changes to a high state and the binary output of the two stages reads 11.

Additional stages can be added in much the same way we just added the second stage. Each new stage becomes the least significant bit of the digitizer and its reference voltage is a weighted sum of the outputs of the more significant stages. Using 5% resistors, the scheme can be carried to 5 bits. 1%'ers would probably take us up to 6 bit resolution; 7 or 8 bit resolution should be realizable by going to active summing amps instead of the passive summing we've used. But, then you're back to complicated again.

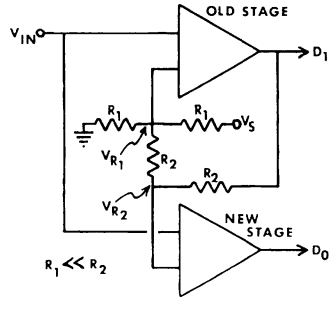
Instead, we'll stop at an easily obtainable 4 bits with the design shown in figure 7. Since the MC3302P is a quad comparator, only one IC is used in this circuit. Like I said, it's simple. Resistors R5, R12, R19, and R27 have been added to give just the slightest hysteresis (positive feedback) to each stage to help overcome any uncertainty at input voltages that correspond exactly to 'change of state' points. When powered from a computer's 5 volt supply, the range of input voltages is also 0 to 5 volts and the pot to be digitized is hung across the supply as laboriously

7

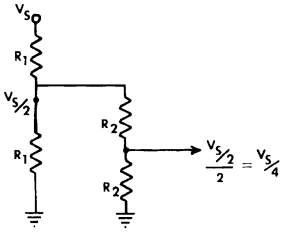
A FOUR BIT DIGITIZER



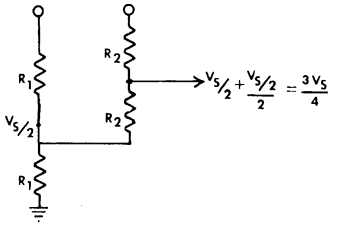
4



5

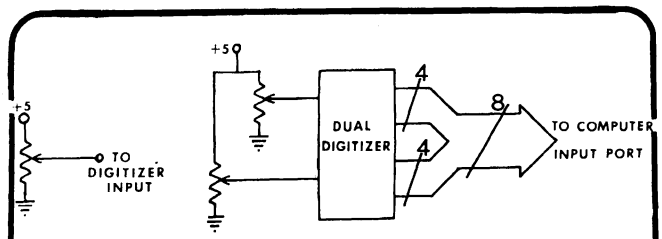


6



depicted in the formidable technical drawing of figure 8. At this point, we may as well establish the standard that the pot should be wired so clockwise rotation of the control causes the output of the digitizer to go from $0 to $F (see test program).

I believe that the most useful configuration for this circuitry is actually two digitizers on a single board, with each half providing half of an 8 bit word. The configuration shown in figure 9 is Paia's EK-7 and is made to plug directly into input port #2 of a Paia 8700 computer. It can also be connected to any 8 bit input port of any computer.



8

9

software consideration

The nicest thing about the digitizer is that it is easy to program for. There are no clocks to worry about and no elaborate software overhead (in fact, none at all). You just read the port to which the digitizer is connected to find the state of the knobs.

A good first example is the short program written for an 8700 to test the unit's operation shown in Listing 1. This program reads the output of the digitizer and shows the result in the 8700's displays. When the value of either of the digitizer outputs changes, the beeper sounds. As the knobs are rotated, the displays should show that the output increases or decreases sequentially without skipping any of the hexadecimal numerals $0 - $F and that there is no interaction between the two digitizer sections.

```
                0010 :TEST FOR 4 BIT DIGITIZER
                0020 :DIGITIZER INPUTS TO PORT #2
                0030 :
                0040 : A) SHOW OUTPUT OF DIGITIZER
                0050 : B) BEEP WHEN VALUE CHANGES
                0130 :
1000- AD 08 08  0140 STAR LDA INPT     :GET DIGIT
1003- C5 20     0160      CMP *TEMP    :SAME AS LAST?
1005- F0 04     0170      BEQ LP1      :YES-BRANCH
1007- 18        0180      CLC          :PREPARE
1008- 20 22 0F  0190      JSR BEEP     :AND BEEP
100B- 85 20     0200 LP1  STA *TEMP    :SAVE VALUE
100D- 8D 20 08  0210      STA DISP     :SHOW VALUE
1010- 4C 00 10  0220      JMP STAR     :AND SO ON....
                0230 :
                0240 :
                0250 END  .EN
```

list1

The fact that there are two digitizer sections on the EK-7, one contributing the upper half-byte and the other the lower half-byte is going to be of great significance in some future software and hardware that we'll be doing.

For now, we'll use the PINK TUNES software (Polyphony July/August 78, pp. 22-26) as an example. When you review PINK TUNES, you'll notice that the statistical properties of the note durations (half notes, quarter notes, dotted notes, etc.) are controlled by the upper half-byte (UHB) and lower half byte (LHB) of memory locations we call MASK and TIME. We don't have the space here to duplicate the detailed explanation of how these variables interact which appeared in Polyphony, and is reprinted in "Friendly Stories About Computers/Synthesizers"; but briefly, both UHBs interact to determine the probability of a dotted note. The LHB of TIME sets the minimum note duration that will occur, while the MASK's LHB controls the range of possible note durations.

These dual half-byte control words are just right for use with a dual half-byte digitizer. From a programming standpoint, all we have to do is read the memory location where the knobs are ($808 on an 8700)

```
                0010 :                KNOBS FOR PINK TUNES
                0080 :
                0090      .OR 1066
                0100 :
                0110 :BEFORE WE BEGIN, NOTE THAT THE FOLLOWING SECTION REPLACES PART OF THE
                0120 :EXISTING PINK TUNES PROGRAM.  PRIMARILY, WE CHANGE THE BRANCH DESTINATION
                0130 :FOR THE BNE AT LOCATION $1066 SO THAT THE BRANCH IS TO THE TESTS WHICH
                0140 :FOLLOW RATHER THAN BACK TO THE START OF THE PROGRAM AS IT WAS ORIGINALLY.
                0150 :
1066- D0 07     0160      BNE TST5     :RATHER THAN TO LP0 AS ORIGINALLY WRITTEN
1068- 20 71 11  0170      JSR SET
1068- 20 2B 1D  0180      JSR NOTE
106E- 00        0190      BRK
                0200 :
                0210 :AS WE JOIN OUR PROGRAM, TEST HAVE ALREADY BEEN MADE TO SEE IF COMMAND
                0220 :FROM KEYBOARD WAS FOR SCRAMBLE, TUNE, OR STOP.  NOW WE ADD TESTS FOR
                0230 :CHANGE TEMPO OR TIME AND MASK PARAMETERS
                0240 :
106F- C9 0C     0250 TST5 CMP 0C       :IS THERE A COMMAND AT ALL?
1071- B0 9B     0260      BCS LP0      :NO, JUST GO AHEAD AND BRANCH TO KEEP ON TRUCKIN'
1073- E9 03     0270      SBC 03       :NORMALIZE COMMAND FOR POINTER USE (CARRY WAS CLEAR)
1075- 08        0280      PHP          :SAVE THE + OR - STATUS OF THE SUBTRACTION FOR LATER
1076- AA        0290      TAX          :AND TRANSFER THE RESULT TO POINTER, MAY USE
1077- AD 00 18  0300      LDA DGIT     :GET THE DIGITIZER OUTPUT
107A- 28        0310      PLP          :NOW RECOVER THE + OR - STATUS OF THAT SUBRACTION
107B- 10 06     0320      BPL TST6     :IF THE POINTER IS =>0 BRANCH TO CHANGE MASK OR TIME
107D- 09 F0     0330      ORA 0F0      :TEMPO CHANGE, SET ALL UHB BITS TO 1'S WITH THIS MASK
107F- 85 A9     0340      STA *TMPO    :THEN SAVE RESULT AS TEMPO CHANGE
1081- D0 8B     0350      BNE LP0      :AND BRANCH ALWAYS TO CONTINUE
1083- 95 88     0360 TST6 STA *TIME,X  :CHANGE TIME OR MASK PARAMETERS
1085- 10 87     0370      BPL LP0      :BRANCH ALWAYS TO KEEP ON
                0380 :
                0390      .EN
```

list2

and put the result in the memory location where PINK TUNES is going to look for the variable that we're changing.

Now, there's a minor difficulty as we have 8 bytes worth of variables (MASK and TIME for each of 4 channels) to set with only two knobs. At some point in the future we'll look at hardware ways to multiplex our digitizer so it can be fed by multiple addressable pots (something like QuASHes in reverse), but for now we're going to actually multiplex the knobs - with software.

Depending on which command pad is being touched, a knob may be controlling minimum note duration on channel A, or the range of durations on channel C, or any of the other possibilities. The program shown in Listing 2 can be added to PINK TUNES to make all this happen. With the software running, the first 12 pads of

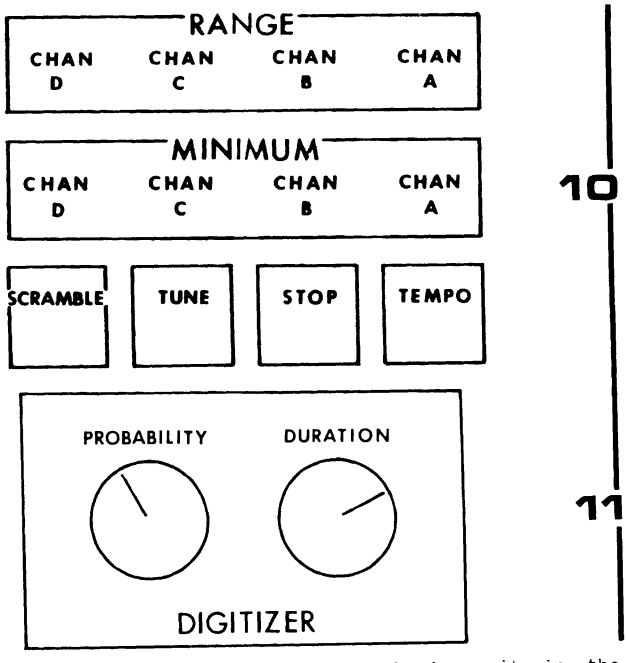
the 8700's command keyboard take on the responsibilities depicted in figure 10.

The first three keys on the computer serve the same function they did in the un-altered PINK TUNES, but from there on it's all new. When TEMPO is touched, the knob corresponding to the LHB of the digitizer provides a coarse control of tempo; the other control has no effect. Touching one of the pads $4 - $B causes the selected parameter for the selected channel to be read from the pots. By the way, thinking of the pots as being labeled as shown in figure 11 will help you keep their functions straight in your mind (particularly if you remember that TEMPO is a duration function).

Yep, the knobs are definitely a plus for PINK TUNES. You can really try things out fast without having to shut everything down and scratch your head each time you want to change a channel from quarter to half notes, and so on. Also, the first program is a good example of how to program the knobs when you're setting variables that are organized as 4 bits each, two to the byte.

But there are other ways that the knobs can be programmed. For example, some parameters simply require more resolution than the 16 quantizing levels that 4 bits provide. An obvious answer is to think of the two knobs as both controlling one value, in which case the UHB knob can be thought of as a coarse range control while the LHB knob is fine tuning (our first test program can be thought of as acting this way). We'll look at another way that resolution can be extended in a moment.

In some cases the 16 quantizing levels provided by a single digitizer "channel" is sufficient resolution, but the resulting parameter must have a greater range than 4 bits allow. A brute force method of dealing with this is to use the output of the digitizer as a pointer to a table of parameter values like the code in Listing 3. This program reads a value from the table based on the setting of the LHB knob and shows it in the displays. In this case the table is an approximation of 1/4 cycle of a sine wave, but it could be anything.



10

11

In some cases the digitizer's output can be used in some way to calculate the parameter value.

One of the difficulties with software multiplexing of the knobs is that unless you're one of those people blessed with eidetic memory you have little way to know what the position of the knob was the last time you set it. In some cases this isn't important, but in others (when you want to smoothly change a parameter from what it is to what you want it to be) it can cause problems. You punch in to change a value and the value immediately jumps to correspond to the current setting of the control. Glitch-ville.

A solution is to use the knob not to set the parameter, but to change it. That may not sound like a big difference, but it is. Using the knob to change the parameter means that when a function is punched in, the current setting of the knob is not important. As the knob is turned, though, the change in its position produces a corresponding change in the parameter. Try running the software in Listing 4.

With the code operating, any changes in the setting of the LHB knob are ignored completely until the parameter change is called for by touching the "0" command pad. Then, as the knob is rotated clockwise, the parameter (as shown in the displays) increases. Unlike the other code that we've examined, when the end of control rotation is reached, you can release the command pad, turn the knob fully counterclockwise, touch the pad again and continue increasing the parameter. This technique not only provides smooth control over a value without having to know its current state, it also extends the range of values that can be set with the knob.

The things that we've covered here are not all the possibilities, but hopefully they will get you started in adding variables to your software. It's really hard to beat a knob.

The following are available from Paia Electronics, Inc., 1020 W. Wilshire, Oklahoma City, OK 73116:
** EK-7 Dual Digitizer kit, with PC board and all parts (including pots, knobs, and sockets); $14.95 + $1 postage and handling.
** "Friendly Stories About Computers/Synthesizers", a compendium of past Lab Notes from Polyphony; $5.00 ppd.

```
                        !L
                        0010 :                    TABLE LOOK-UP DEMO
                        0020 :
                        0030 :THE DIGITIZER IS READ AND THE LHB IS MASKED OFF WITH AN 'AND'.  THE
                        0040 :RESULT IS PLACED IN THE X REGISTER FOR USE AS A POINTER TO THE TABLE OF
                        0045 :VALUES WHICH OCCUPIES THE MEMORY IMMEDIATELY FOLLOWING THE PROGRAM.
                        0050 :THE VALUE CORRESPONDING TO THE KNOB POSITION IS FETCHED AND SHOWN IN
                        0060 :THE DISPLAYS.
                        0070 :
1000-  AD 08 18         0080 STAR LDA DGIT     :READ THE DIGITIZER
1003-  29 0F            0090      AND 0F       :'AND' WITH MASK (00001111) TO MAKE UHB ZERO
1005-  AA               0100      TAX          :THEN PUT TO X REGISTER TO USE AS POINTER
1006-  B5 0E            0110      LDA *TABL,X  :GET THE PARAMETER VALUE FOR THIS KNOB SETTING
1008-  8D 20 18         0120      STA DISP     :SHOW THE PARAMETER VALUE
100B-  4C 00 10         0130      JMP STAR     :THEN LOOP FOR MORE
                        0140 :
                        0150 TABL .HS 0019324A62788E82B4C5D4E1ECF4FAFE
                        0160 :
                        0170      .EN
```

list 3

list 4

```
                        0010 :                    DELTA TUNE DEMO
                        0020 :
                        0030 :AFTER A SHORT DELAY GENERATED BY CALLING THE MUS 1.0 SUBROUTINE LOOK,
                        0040 :WE READ THE COMMAND KEYBOARD BY CALLING THE MONITOR SUBROUTINE DECODE.
                        0050 :ON RETURNING FROM THIS SUBROUTINE THE ACCUMULATOR AND Y REGISTER CONTAIN
                        0060 :THE NUMBER OF THE LOWEST KEY THAT WAS PRESSED ($18 FOR NO KEY).  THE
                        0070 :CARRY FLAG IS CLEARED BY DECODE IF THE KEY WAS TOUCHED THIS SCAN BUT NOT
                        0080 :THE LAST, I.E.  IF THE KEY WAS JUST TOUCHED
                        0090 :
1000-  20 4E 1D         0100 DLTA JSR LOOK     :THE CAPACITIVE KEYBOARD REQUIRES A DELAY BETWEEN SCANS
1003-  20 00 1F         0110      JSR DECD     :READ THE COMMAND KEYBOARD
1006-  D0 F8            0120      BNE DLTA     :IF ZERO KEY NOT TOUCHED, LOOP
1008-  AD 08 18         0130      LDA DGIT     :CHANGE COMMAND ASSERTED, SO GET THE DIGITIZER OUTPUT
                        0140 :
                        0150 :NOW THE ACCUMULATOR HAS THE DIGITIZED KNOB POSITION.  WE'RE REALLY ONLY
                        0160 :INTERESTED IN THE LHB, SO WE MAKE THE UHB ZERO WITH AN 'AND'.  IF THE
                        0170 :COMMAND WAS JUST ASSERTED, WE SKIP THE CALCULATION OF CHANGE IN SETTING
                        0180 :AND SIMPLY SAVE THE CURRENT SETTING AS THE STARTING VALUE
                        0190 :
100B-  29 0F            0200      AND 0F       :'AND' WITH MASK TO MAKE UHB ZERO
100D-  90 17            0210      BCC DNX0     :IF COMMAND JUST ASSERTED, SKIP CALCULATING CHANGE
100F-  48               0220      PHA          :SAVE KNOB POSITION ON THE STACK FOR USE LATER
1010-  38               0230      SEC          :PREPARE FOR SUBTRACTION TO FOLLOW
                        0240 :
                        0250 :IT'S TIME TO SEE HOW THE KNOB HAS CHANGED.  CURRENT SETTING IS SUBTRACTED
                        0260 :FROM PREVIOUS SETTING AND THE DIFFERENCE (MAY BE + OR -) IS ADDED TO THE
                        0270 :CURRENT VALUE OF THE PARAMETER.  TESTS ARE MADE TO SEE THAT WE'RE WITHIN
                        0280 :THE ARBITRARY RANGE $00-$3F AND IF OUT OF RANGE THE LIMIT IS SUBSTITUTED
                        0290 :FOR THE CURRENT PARAMETER
                        0300 :
1011-  E5 00            0310      SBC *TEMP    :TEMP IS THE POSITION OF THE KNOB THE LAST TIME THROUGH
1013-  18               0320      CLC          :NOW PREPARE FOR ADDITION
1014-  65 01            0330      ADC *PARM    :ADD THE DIFFERENCE BETWEEN NOW AND LAST TIME TO VALUE
1016-  10 02            0340      BPL DNX1     :IF GREATER THAN ZERO, SKIP THE NEXT INSTRUCTION
1018-  A9 00            0350      LDA 00       :IF WE'RE HERE, WE'RE UNDER-RANGE.  MAKE PARAMETER ZERO
101A-  C9 3F            0360 DNX1 CMP 3F       :ARE WE GREATER THAN THE MAX ALLOWABLE FOR PARAMETER?
101C-  90 02            0370      BCC DNX2     :NO, SO BRANCH TO SKIP NEXT INSTRUCTION
101E-  A9 3F            0380      LDA 3F       :OVER-RANGE, MAKE PARAMETER EQUAL TO MAX LIMIT
1020-  85 01            0390 DNX2 STA *PARM    :SAVE THE NEW VALUE OF THE PARAMETER
1022-  8D 20 18         0400      STA DISP     :AND SHOW IT IN THE DISPLAYS
                        0410 :
                        0420 :NOW WE GET READY FOR THE NEXT PASS BY SAVING THE CURRENT KNOB POSITION
                        0430 :
1025-  68               0440      PLA          :PULL THE DIGITIZER OUTPUT FROM THE STACK
1026-  85 00            0450 DNX0 STA *TEMP    :AND SAVE IT TO DETERMINE CHANGE IN SETTING NEXT TIME
1028-  4C 00 10         0460      JMP DLTA     :THEN JUMP TO START TO CONTINUE
                        0470 :
                        0480      .EN
```

POLY-SPLIT

Many times we've talked about how the personality of our computer based equipment is a function of the operating system software that we happen to be running at the moment. Let's play some head games with the gear and feed it some code that will give it a split personality.

POLY-SPLIT does just that; it gives us two complete polyphonic synthesizer systems under the control of one keyboard. Play a chord or note on the lower keys and they are always assigned to a lower group of outputs. Play on higher keys and the result is assigned to another group.

Before we get into the listing of this program and its operation and use, we need to keep one fact clearly in mind; POLY-SPLIT is simply an extension of the polyphonic personality offered in MUS 1.0. All of the options offered by that code (STGs, dynamic output refresh, etc.) are provided by this one also. Since many of MUS 1.0's subroutines are used by POLY-SPLIT you must have this PROM or its equivalent available, and the variables that you manually initialize for MUS 1.0 (see LAB NOTES: MUS 1.0, April/ May 1978 Polyphony) must be set for POLY-SPLIT also.

In addition to OUTS, CTRL, etc. which MUS 1.0 used there is a new variable which is unique to POLY-SPLIT; OUT2 ($BF). This is the variable that tells the program how many channels are to be set aside for use exclusively by AGO keys below the split point. Notice specifically that if MUS 1.0's STG option is selected, the number entered into this variable must include those channels which will be producing envelope transients. (i.e. The number entered for OUT2 will always be an even number when STGs are being used.)

For example, if you have hardware (QuASH, etc.) for eight channels, this number is entered into the normal MUS 1.0 location for it; OUTS ($EA). If you want to split these into three channels for low keys and five for high keys, you would set OUT2 ($BF) to contain 03.

The program appears at the end of this column and is loaded starting at location $000 in the same way that we've loaded programs in the past. If you're the careful sort, you will also save the program on tape as soon as it's loaded so that if there's a problem it won't wipe out all of your work.

When the program has been loaded, preset the MUS 1.0 variables according to your preferences and application, and set the low channels variable (OUT2) as discussed above.

Run the program from location $000. With POLY-SPLIT running, keys 0 and 1 on the command keyboard retain the functions that they had under MUS 1.0. Key 0 clears and mutes the system; key 1 causes all of the channels to produce a note corresponding to middle C on the AGO keyboard.

A use for command key 2 has now been added; it provides a means of changing the split point while you're playing. Touch this pad and, as long as it's held down, any key on the AGO keyboard that you press will become the new split point. Now while playing, any key below the split point will be assigned to the channels that you've set aside for them, while keys greater or equal to the split point will be assigned to the remaining channels.

```
                0010 :*********************************
                0020 :*                               *
                0030 :*         POLY-SPLIT            *
                0040 :*                               *
                0050 :*   A PROGRAM FOR POLYPHONIC    *
                0060 :*       SPLIT KEYBOARD          *
                0070 :*                               *
                0080 :*             BY                *
                0090 :*        JOHN SIMONTON          *
                0100 :*                               *
                0110 :* (C) 1979 - PAIA ELECTRONICS    *
                0120 :*                               *
                0130 :*********************************
                 0010 KTBL .DL 00E0
                 0020 NTBL .DL 00D8
                 0030 HKEY .DL 00A2
                 0040 SPLT .DL 00A1
                 0050 OUT2 .DL 00EC
                 0060 OUTT .DL 00EB
                 0070 OUTS .DL 00EA
                 0080 TRGN .DL 0DC3
                 0090 INIT .DL 0D21
                 0100 NOTE .DL 0D2B
                 0110 POLY .DL 0D71
                 0120 DECD .DL 0F00
                 0130 :
                 0140 :FIRST, SYSTEM THINGS ARE DISPOSED OF.  THE SYSTEM IS
                 0150 :INITIALIZED USING MUS 1.0'S "INIT" ROUTINE, THEN THE
                 0160 :QUASH CHANNELS ARE REFRESHED AND THE AGO KEYBOARD
                 0170 :SCANNED ALSO USING ROUTINES FROM MUS 1.0
                 0180 :FINALLY, THE PIEBUG ROUTINE "DECODE" IS USED TO READ THE
                 0190 :COMMAND KEYBOARD AND ANY COMMANDS ARE EXECUTED.
                 0200 :0-SYSTEM CLEAR AND RE-INIT; 1-TUNE ALL CHANNELS;
                 0210 :2-SET SPLIT POINT, ANY AGO KEY PRESSED BECOMES SPLIT
                 0220 :
                 0230      .OR 1000
                 0240 :
1000-  A5 EB     0250 STAR LDA *OUTT    :GET THE # OF RESERVED LOW CHANS
1002-  85 EC     0260      STA *OUT2    :SAVE PERMANENTLY
1004-  A2 07     0270 POSP LDX 07       :SET UP A POINTER/COUNTER
1006-  A9 00     0280 SLP9 LDA 00       :AND GET READY TO ZERO STUFF
1008-  95 A2     0290      STA *HKEY,X  :ZERO THE TEMPORARY BUFFER
100A-  CA        0300      DEX          :AND POINT TO THE NEXT
100B-  10 F9     0310      BPL SLP9     :IF SOME ARE LEFT, LOOP
100D-  20 21 0D  0320      JSR INIT     :MUS 1.0 - INTIALIZE SYSTEM
1010-  20 2B 0D  0330 SLP6 JSR NOTE     :MUS 1.0 - REFRESH AND READ AGO KBD
1013-  20 00 0F  0340      JSR DECD     :PIEBUG - READ COMMAND KEYBOARD
1016-  F0 EC     0350      BEQ POSP     :IF COMMAND = 0, BRANCH TO RE-INIT
1018-  C9 01     0360      CMP 01       :IS COMMAND = 1?
101A-  D0 07     0370      BNE NTST     :NO, BRANCH TO NEXT TEST
101C-  A9 2E     0380      LDA 2E       :WILL BECOME MIDDLE C
101E-  20 23 0D  0390      JSR INIT+02  :USE PART OF MUS 1.0 INITIALIZE
1021-  F0 ED     0400      BEQ SLP6     :BRANCH ALWAYS
1023-  C9 02     0410 NTST CMP 02       :IS COMMAND = 2?
1025-  D0 08     0420      BNE SPLI     :NO, BRANCH TO POLY-SPLIT PROGRAM
1027-  A5 E7     0430      LDA *KTBL+07 :GET THE LOWEST KEY DOWN
1029-  F0 E5     0440      BEQ SLP6     :IF NONE ARE DOWN, LOOP
102B-  85 A1     0450      STA *SPLT    :SAVE THE KEY AS THE SPLIT POINT
102D-  D0 E1     0460      BNE SLP6     :BRANCH ALWAYS
```

```
                        0470  :
                        0480  :NOW THE SPLIT PROGRAM.  AT THIS POINT A LIST OF THE
                        0490  :AGO KEYS WHICH THE MUS 1.0 SUBROUTINE "LOOK" FOUND TO
                        0500  :BE PRESSED HAS BEEN COMPILED AND SAVED IN THE INPUT BUFFER
                        0510  :AREA "KTBL".  WE BEGIN BY REMOVING FROM THE INPUT BUFFER
                        0520  :ALL THOSE KEYS WHICH ARE ABOVE THE SPLIT POINT AND
                        0530  :TRANSFERING THEM TO THE TEMPORARY BUFFER AREA "HKEY".
                        0540  :
102F-   A0 07          0550  SPLI LDY 07       :SET UP POINTER TO HIGH BUFFER
1031-   A2 07          0560       LDX 07       :AND ONE TO INPUT BUFFER
1033-   B5 E0          0570  SLP0 LDA *KTBL,X  :GET THE KEY
1035-   F0 0F          0580       BEQ SNX1     :IF ZERO, GO TO NEXT
1037-   C5 A1          0590       CMP *SPLT    :GREATER THAN SPLIT POINT?
1039-   90 08          0600       BCC SNX0     :IF NOT GREATER, BRANCH
103B-   99 A2 00       0610       STA HKEY,Y   :GREATER, SAVE IN HIGH BUFFER
103E-   88             0620       DEY          :POINT TO NEXT HIGH KEY BUFFER
103F-   A9 00          0630       LDA 00       :PREPARE AND
1041-   95 E0          0640       STA *KTBL,X  :ZERO THIS KEY
1043-   CA             0650  SNX0 DEX          :POINT TO NEXT KEY
1044-   10 ED          0660       BPL SLP0     :IF SOME LEFT, LOOP
                        0670  :
                        0680  :NEXT THE NUMBER OF CHANNELS AVAILABLE FOR LOW KEY USE
                        0690  :IS TRANSFERRED TO THE TEMPORARY COUNTER "OUTT" AND THE
                        0700  :MUS 1.0 ALLOCATION PROGRAM POLY IS CALLED TO ASSIGN LOW
                        0710  :KEYS TO LOW CHANNELS.
                        0720  :
1046-   A5 EC          0730  SNX1 LDA *OUT2    :GET THE NUMBER OF LOW CHANS AVAILABLE
1048-   85 EB          0740       STA *OUTT    :AND PUT IT IN THE TEMPORARY COUNTER
104A-   20 75 0D       0750       JSR POLY+04  :AND CALL THE MAIN PORTION OF POLY
                        0760  :
                        0770  :NOW THAT THE LOW KEYS HAVE BEEN ALLOCATED TO LOW CHANNELS,
                        0780  :THE HIGH KEYS ARE TAKEN FROM "HKEY" AND PLACED BACK IN THE
                        0790  :INPUT BUFFER (KEYS ALREADY ALLOCATED ARE REMOVED FROM THE
                        0800  :INPUT BUFFER).  SIMULTANEOUSLY THE LOW CHANNELS ARE MOVED
                        0810  :TO HKEY AND ALL LOW CHANNELS IN THE OUTPUT BUFFER
                        0820  :ARE MARKED AS "IN USE" SO THAT THEY WILL BE IGNORED
                        0830  :WHEN HIGH KEYS ARE ALLOCATED.
                        0840  :
104D-   A4 EC          0850       LDY *OUT2    :A COUNTER TO MOVE ONLY THE LOW CHANNELS
104F-   A2 07          0860       LDX 07       :AND A POINTER/COUNTER
1051-   B5 A2          0870  SLP1 LDA *HKEY,X  :GET THE HIGH KEY FROM TEMP BUFFER
1053-   95 E0          0880       STA *KTBL,X  :PUT IT IN THE INPUT BUFFER
1055-   88             0890       DEY          :ONE LESS LOW CHANNEL  TO DO
1056-   30 08          0900       BMI SNX2     :ALL LOW CHANNELS DONE, BRANCH
1058-   B5 D8          0910       LDA *NTBL,X  :GET THE LOW NOTE
105A-   95 A2          0920       STA *HKEY,X  :PUT IT IN TEMPORARY BUFFER
105C-   09 40          0930       ORA 40       :THEN SET THE TRIGGER TO MARK NOTE
105E-   95 D8          0940       STA *NTBL,X  :AND REPLACE THE NOTE
1060-   CA             0950  SNX2 DEX          :ONE LESS CHANNEL, POINT TO NEXT
1061-   10 EE          0960       BPL SLP1     :IF SOME LEFT, LOOP
                        0970  :
                        0980  :NOW POLY IS CALLED AGAIN, THIS TIME TO ALLOCATE HIGH CHANNELS
                        0990  :
1063-   38             1000       SEC          :PREPARE FOR SUBTRACTION
1064-   A9 10          1010       LDA 10       :16 CHANNELS SUPPORTED BY MUS1
1066-   E5 EC          1020       SBC *OUT2    :LESS THE LOW RESERVED CHANNELS
1068-   AA             1030       TAX          :RESULT IS POINTER
1069-   38             1040       SEC          :ANOTHER SUBTRACTION - PREPARE
106A-   A5 EA          1050       LDA *OUTS    :TOTAL HARDWARE CHANNELS
106C-   E5 EC          1060       SBC *OUT2    :LESS LOW RESERVED CHANNELS
106E-   85 EB          1070       STA *OUTT    :BECOMES CHANNELS LEFT TO ALLOCATE
1070-   20 77 0D       1080       JSR POLY+06  :CALL MAJOR PORTION OF POLY
                        1090  :
                        1100  :FINALLY, THE REAL STATE OF THE LOW CHANNELS IS RESTORED
                        1110  :TO THE OUTPUT BUFFER.  SIMULTANEOUSLY THE TEMPORARY BUFFER
                        1120  :IS ZERO'D FOR THE NEXT PASS.
                        1130  :
1073-   A4 EC          1140       LDY *OUT2    :NUMBER OF LOW CHANNELS FOR COUNTER
1075-   A2 07          1150       LDX 07       :POINTER/COUNTER
1077-   88             1160  SLP2 DEY          :ONE LESS LOW CHANNEL
1078-   30 04          1170       BMI SNX3     :AND IF ALL DONE, SKIP NEXT TRANSFER
107A-   B5 A2          1180       LDA *HKEY,X  :GET THE REAL CHANNEL STATE
107C-   95 D8          1190       STA *NTBL,X  :PLACE IN OUTPUT BUFFER
107E-   A9 00          1200  SNX3 LDA 00       :NOW GET READY AND
1080-   95 A2          1210       STA *HKEY,X  :ZERO THIS TEMPORARY BUFFER LOCATION
1082-   CA             1220       DEX          :ONE LESS TEMP BUFFER LOCATION
1083-   10 F2          1230       BPL SLP2     :IF SOME REMAIN, LOOP
1085-   30 89          1240       BMI SLP6     :BRANCH ALWAYS TO CONTINUE
                        1250  :
                        1260  END  .EN
```

```
1280  ***********************************
1290  *           NOTES:                *
1300  *                                 *
1310  *  DUMP PROGRAM FROM 0000-0090    *
1320  *                                 *
1330  *  SET THESE LOCATIONS:           *
1340  *                                 *
1350  *     $0E8  CTRL $40  DYNAMIC     *
1360  *     $0E9  ODLY $20  DELAY       *
1370  *     $0EA  OUTS $XX  TOT CHANS    *
1380  *     $0EB  OUTT $XX  LOW CHANS    *
1390  *                                 *
1400  *  COLD START - $0000              *
1410  *  WARM START - $0004              *
1420  *                                 *
1430  *                                 *
1440  *  NOTE THE FOLLOWING THINGS:     *
1450  *                                 *
1460  * 1) THE PROGRAM IS RELOCATABLE;  *
1470  *    IT MAY BE LOADED AND RUN IN  *
1480  *    ANY NON-CONFLICTING MEMRORY  *
1490  *    SPACE                        *
1500  *                                 *
1510  * 2) CALLING POLY TWICE IS NOT    *
1520  *    EXTRA EFFICIENT.  TIME RE-   *
1530  *    QUIREMENTS DICTATE MEDIUM    *
1540  *    TEMPO KNOB SETTING - ABOUT   *
1550  *    10 MS/SCAN                   *
1560  *                                 *
1570  * 3) AS SOON AS THE PROGRAM IS     *
1580  *    RUNNING, TOUCH COMMAND PAD   *
1590  *    2 AND THE KEY WHICH IS TO    *
1600  *    BE THE SPLIT POINT.  THEN 1  *
1610  *    TO TUNE AND FINALLY 0        *
1620  *    BEFORE PLAYING               *
1630  *                                 *
1640  ***********************************
1650  POLY-SPLIT 8.8
```

OG93: AN INTERPRETIVE ARPEGGIATION PROGRAMMER & EDITOR

One of the major advantages that our hybrid computer/synthesizer system offers is the ability to realize a class of new tricks which for lack of a better term we'll call "keyboard effects". I have in mind new sounds which arise not so much from the timbre of each note, but from the timing and sequence in which the keys played are converted to notes and how they're allocated to available output channels.

Using this definition, I suppose that POLY-SPLIT from last time would qualify as a keyboard effect because it affects the way that keys held down are allocated to note-producing output channels. But, ECHO (January-March 1979 Polyphony, page 29) is more specificly what I feel the term should mean because with that program new effects (and at short delay settings, new timbres) arise that would be extremely difficult to accomplish without some means of juggling key activations and how they're assigned to outputs.

Another good example would be the ORGASMATRONIC GLIDE arpeggiation trick that the keyboard encoder and D/A did by themselves (remember?). Hold down a bunch of keys and the encoder, while scanning, stopped momentarily when it reached one of the down keys and played the note briefly before continuing the scan. When another key was found down, it stopped again to play that one, and so on. Altogether an alright thing that allowed arpeggiations to be played much more rapidly than they could be without electronic assistance.

When we installed the computer in the loop, we lost Orgasmatronic Glide (OG), which maybe was not such a huge sacrifice when considering the power that was gained in the process; but still, I know several folks who mourned the loss because it was an effect that they were using to good purpose in their music.

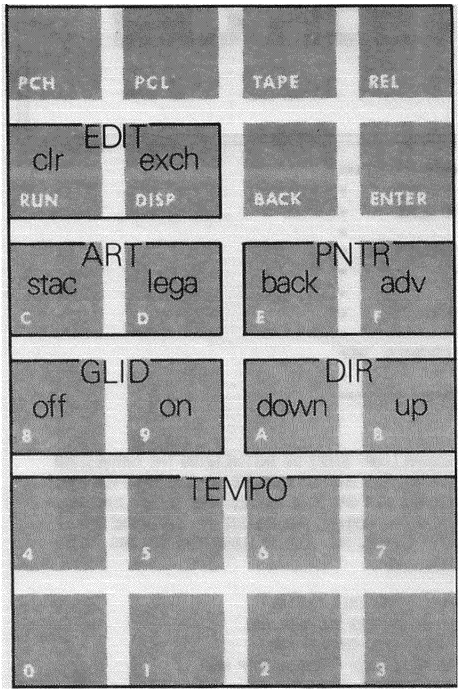
Here's a terrific replacement. This new program does the same thing that the old OG did, hold down a bunch of keys and it plays them in sequence, but it also gives control that wasn't possible with the old "state machine" version. For instance, it can arpeggiate down-scale as well as up. And it plays staccato or legato. It also allows touch pad control of glide and similar control of the tempo of the arpeggiation.

Great. But not the greatest part, we'll get to that soon.

```
                                0010 :*****************************
                                0020 :*                           *
                                0030 :*    ORGASMATRONIC GLIDE    *
                                0040 :*                           *
                                0050 :* ARPEGIATION PROGRAMMER AND *
                                0060 :*          EDITOR           *
                                0070 :*            BY             *
                                0080 :*    JOHN S. SIMONTON, JR   *
                                0090 :*                           *
                                0100 :*(C) 1979 PAIA ELECTRONICS, INC*
                                0110 :*                           *
                                0120 :*****************************
                                0130 :
                                0140 :
                                0540 :THIS IS THE MAIN PROGRAM LOOP. START BY INITIALIZING THE SYNTHESIZER
                                0550 :AND CALLING THE QUASH DRIVERS AND AGO KBD READING ROUTINES FROM MUS1
                                0560 :CHECK TO SEE IF A COMMAND KEY HAS BEEN TOUCHED; AND IF SO, JUMP TO
                                0570 :SUBROUTINE TO DETERMINE THE COMMAND AND EXECUTE IT. DETERMINE THE
                                0580 :POINTER FOR THE OUTPUT CHANNEL AND JUMP TO SUBROUTINE FOR ORG. GLIDE
                                0590 :PROCESSING. ON RETURN, LOOP.
*1000LLL                        0600 :
1000-  20 21 1D                 0610      JSR INIT      :MUS1 SYNTH INIT ROUTINE
1003-  20 2B 1D                 0620 LOOP JSR NOTE      :QUASH DRIVERS AND READ AGO
1006-  20 00 1F                 0630      JSR DECD      :PIEBUG READ COMMAND KBD
1009-  B0 03                    0640      BCS HERE      :IF NO NEW KEY TOUCHED, SKIP NEXT
1008-  20 00 11                 0650      JSR CMND      :CALL COMMAND DECODER
100E-  A0 0F                    0660 HERE LDY 0F        :POINTER TO ORG. GLIDE OUTPUT CHANNEL
1010-  20 16 10                 0670      JSR STAR      :CALL ORG. GLIDE PROGRAM
1013-  4C 03 10                 0680      JMP LOOP      :LOOP TO CONTINUE
                                0690 :
                                0700 :FIRST THE TIMER IS TESTED AND IF NOT TIME FOR THE NEXT NOTE TO BE
                                0710 :PROCESSED THE STACCATO CONTROL BIT IS CHECKED AND IF CLEAR
                                0720 :(STACCATO) BRANCH IS TAKEN TO DE-TRIGGER NOTE IN OUTPUT
                                0730 :BUFFER. IF LEGATO MODE, EXIT IS IMMEDIATE
                                0740 :
1016-  24 E7                    0750 STAR BIT *KTBL+07 :ARE THERE ANY AGO KEYS DOWN?
1018-  50 1E                    0760      BVC SINT      :NO KEYS, BRANCH TO RE-INIT ARP. POINTER
101A-  C6 72                    0770      DEC *TIMR     :OTHERWISE, DECREMENT THE TIMER
101C-  30 05                    0780      BMI ADVA      :IF EVENT TIME, BRANCH
101E-  24 74                    0790      BIT *SCTL     :OTHERWISE CHECK FOR STACCATO AND IF TRUE....
1020-  50 46                    0800      BVC CLRN      :BRANCH TO CLEAR TRIGGER FROM OUTPUT NOTE
1022-  60                       0810      RTS           :OTHERWISE, RETURN WITHOUT CLEARING TRIGGER
                                0820 :
                                0830 :IF IT'S TIME FOR A NOTE TO BE PROCESSED, THE POINTER TO THE INPUT
                                0840 :BUFFER IS ADVANCED (EITHER FORWARD OR BACKWARD) AND IF THERE IS NO
                                0850 :MORE BUFFER LEFT WE DROP THROUGH TO ADVANCE THE POINTER TO THE SEQUENCE
                                0860 :BUFFER TO GET THE NEXT SET OF GLIDE PARAMETERS.  IF WE ARE NOT YET
                                0870 :TO THE END OF THE IN BUFFER, WE BRANCH OUT TO RESET THE TIMER, ETC.
                                0880 :
1023-  A6 73                    0890 ADVA LDX *PNTR     :GET POINTER TO INPUT BUFFER
1025-  24 74                    0900      BIT *SCTL     :CURRENTLY ARPEGGIATING UP?
1027-  10 05                    0910      BPL DOWN      :NO, BRANCH TO DO DOWN
1029-  CA                       0920      DEX           :TO GO UP-SCALE, DECREMENT POINTER
102A-  30 07                    0930      BMI SADV      :IF POINTER NOW <0, BRANCH
102C-  10 26                    0940      BPL STIM      :STILL IN RANGE, BRANCH ALWAYS
102E-  E8                       0950 DOWN INX           :DOWN-SCALE, INCREMENT POINTER
102F-  E0 08                    0960      CPX 08        :OUT OF RANGE?
1031-  D0 21                    0970      BNE STIM      :STILL IN RANGE, BRANCH
                                0980 :
                                0990 :IF WE GET HERE (SADV) IT MEANS THAT WE HAVE PLAYED ALL OF THE KEYS
                                1000 :THAT WERE DOWN AND HAVE REACHED THE END OF THE INPUT BUFFER
                                1010 :NOW IT'S TIME TO GET THE NEXT ENTRY FROM THE CONTROL SEQUENCE.
                                1020 :WE TEST TO SEE IF WE ARE AT THE END OF THE SEQUENCE AND IF SO THE
                                1030 :POINTER IS RE-INITIALIZED. OTHERWISE, THE COMMAND IS FETCHED AND IF
```

Enter the program as outlined at the end of the column and start it running, then press down a group of keys. If you've done everything correctly, you should hear a relatively slow down-scale arpeggiation of the notes that you're holding down. When the lowest note has played, the sequence should start again from the highest.

Now let's play with the control some. Here's what the keys mean with OG93 running:



Touching the DIR:UP pad will cause the arpeggiation to change direction from down-scale to up. GLID:ON turns the glide for the arpeggiation channel on and (you guessed it) GLID:OFF turns it off.

The LEGATO ARTICULATION pad causes the trigger signal to remain high as long as any keys are down so that there will be no re-articulation as one note finishes playing and the next begins. STACCATO ARTICULATION triggers the note the first instant that it plays then releases the trigger.

The TEMPO keys cause the rate of arpeggiation to change from slow (7) to fast (0) over a range from so slow that almost anyone could play the run manually to a rate that's so fast that the sequence begins to take on the texture of a chord (which should give you a clue to one interesting application of OG93 in a piece of music).

If you were an Orgamatronic Glide fan in the first place, we could probably stop here and you'd be completely happy - the program is a lot better than the old manual version. We'd also be stopping before we really got started, because by far the most interesting feature of OG93 is that it's an interpreter that allows us to program a series of arpeggiations and an editor that makes the entry and manipulation of those programs easier.

```
                                1040 :ZERO IT MEANS THAT IT IS THE END OF THE SEQUENCE AND THE POINTER
                                1050 :IS ALSO REINITIALIZED
                                1060 :
1033-  A6 76                    1070 SADV LDX *SPNT    :GET CONTROL SEQUENCE POINTER
1035-  CA                       1080      DEX          :POINT TO NEXT SEQUENCE ENTRY
1036-  10 02                    1090      BPL GSEQ     :IF NOT TO END, BRANCH
1038-  A2 07                    1100 SINT LDX 07       :RE-INIT SEQUENCE POINTER
103A-  86 76                    1110 GSEQ STX *SPNT    :SAVE SEQUENCE POINTER
103C-  B5 77                    1120      LDA *CSEQ,X  :GET COMMAND FROM CONTROL SEQ.
103E-  F0 F8                    1130      BEQ SINT     :ZERO ENDS THE SEQUENCE, BRANCH
                                1140 :
                                1150 :A NEW COMMAND FROM THE SEQUENCE. FIRST USE IT TO SET OR CLEAR THE
                                1160 :THE GLIDE CONTROL BIT FROM THE TRANSPOSE BUFFER.  IN THE PROCESS,
                                1170 :THE NEW COMMAND IS SHIFTED ONE BIT TO THE LEFT; WHICH MULTIPLIES
                                1180 :THE TEMPO VARIABLE BY 2 AND SHIFTS THE UP/DOWN AND LEGA/STACC BITS
                                1190 :INTO MORE EASILY TESTED POSITIONS.
                                1200 :
1040-  85 74                    1210 GLID STA *SCTL    :SAVE SEQUENCE ENTRY IN CONTROL BUFFER
1042-  B9 C0 00                 1220      LDA TTBL,Y   :GET THE CURRENT TRANSOSE BUFFER ENTRY
1045-  2A                       1230      ROL          :ROTATE GLIDE BIT TO CARRY
1046-  06 74                    1240      ASL *SCTL    :ROTATE CONTROL WORD GLIDE TO CARRY
1048-  6A                       1250      ROR          :ROTATE CARRY TO GLIDE BIT
1049-  99 C0 00                 1260      STA TTBL,Y   :THEN RETURN TO TRANSPOSE BUFFER
                                1270 :
                                1280 :THIS LITTLE ROUTINE DETERMINES WHETHER SCAN IS UP OR DOWN AND
                                1290 :INITIALIZES THE POINTER TO THE PROPER VALUE
                                1300 :SKYP-SET KEY POINTER
                                1310 :
104C-  A2 07                    1320 SKYP LDX 07       :PREPARE FOR ARP. UP INITIAL POINTER
104E-  24 74                    1330      BIT *SCTL    :CHECK COMMAND BUFFER - ARP. UP?
1050-  30 02                    1340      BMI STIM     :YES, BRANCH
1052-  A2 00                    1350      LDX 00       :NO, ARP. DOWN INTIAL POINTER
                                1360 :
                                1370 :NOW THE ROUTINE TO RESET THE TIMER. SINCE ALL KEY POINTER MANIPULATIONS
                                1380 :WIND UP AT THIS POINT, THE FIRST INSTRUCTION IS TO SAVE THIS POINTER
                                1390 :THE TIMER VALUE IS EXTRACTED FROM THE CONTROL WORD SCTL
                                1400 :STIM-SET TIMER
                                1410 :
1054-  86 73                    1420 STIM STX *PNTR    :SAVE INPUT BUFFER POINTER
1056-  A9 1F                    1430      LDA 1F       :PREPARE MASK AND
1058-  25 74                    1440      AND *SCTL    :GET THE TIMER (TEMPO) VALUE
105A-  85 72                    1450      STA *TIMR    :AND SAVE IN THE TIMER VARIABLE
                                1460 :
                                1470 :NOW WE GET THE CURRENT NOTE OF INTEREST FROM THE INPUT BUFFER
                                1480 :AND IF THE KEY IS NOT DOWN, A CHECK IS MADE TO SEE IF ANY KEYS
                                1490 :ARE DOWN. IF NONE ARE, THE TIMER IS TRICKED INTO TIMING OUT THE
                                1500 :NEXT TIME THROUGH WHICH WILL THEN RESULT IN THE WHOLE COMMAND
                                1510 :SEQUENCE FOLLOWING SYSTEM BEING RESET
                                1520 :
105C-  B5 E0                    1530      LDA *KTBL,X  :GET THE CURRENT KEY FROM INPUT BUFFER
105E-  D0 0D                    1540      BNE BOUT     :IF ZERO, NO KEY - BRANCH
1060-  24 E7                    1550      BIT *KTBL+07 :ARE ANY KEYS DOWN?
1062-  70 BF                    1560      BVS ADVA     :YES, BRANCH
1064-  A9 01                    1570      LDA 01       :NO, PREPARE TO MAKE TIMER RUN OUT
1066-  85 72                    1580      STA *TIMR    :NEXT PASS THROUGH
1068-  B9 D0 00                 1590 CLRN LDA NTBL,Y   :GET THE CURRENT OUTPUT NOTE
106B-  29 BF                    1600      AND 0BF      :CLEAR THE TRIGGER FLAG
106D-  99 D0 00                 1610 BOUT STA NTBL,Y   :AND REPLACE IN OUTPUT BUFFER
1070-  60                       1620      RTS          :RETURN
                                1630 :
                                1640 :NOW SOME TEMPORARY LOCATIONS AND THEIR INITIAL STATES
                                1650 :
1071  00                        1660 TEMP .HS 00
1072  01                        1670 TIMR .HS 01
1073  08                        1680 PNTR .HS 08
1074  C4                        1690 SCTL .HS C4
1075  07                        1700 PPNT .HS 07
1076  07                        1710 SPNT .HS 07
                                1720 CSEQ .HS 000000000000E404
                                1740      .OR 10E8
                                1750 STUP .HS 402004
                                1770      .OR 1100
                                1790 :THIS IS THE COMMAND KEY DECODING AND SEQUENCE EDITING SUBROUTINE
                                1800 :# OF COMMAND KEY IS IN Y REGISTER
                                1810 :
1100-  A6 75                    1820 CMND LDX *PPNT    :GET THE EDITORS POINTER TO COMMAND SEQ
1102-  B5 77                    1830      LDA *CSEQ,X  :GET THE COMMAND POINTED TO (IN ACC, DON'T FORGET)
1104-  C0 10                    1840      CPY 10       :IS KEY 10 - CLEAR COMMAND SEQUENCE
1106-  F0 0F                    1850      BEQ CLR      :YES, BRANCH
1108-  90 1F                    1860      BCC CNXT     :NO, IT'S LESS THAN "F", BRANCH
                                1870 :
                                1880 :THE KEY IS 11 OR GREATER.  EXCHANGE THE COMMAND POINTED TO WITH
                                1890 :TEMPORARY STORAGE LOCATION TEMP. NOTE THAT THIS CAN BE USED TO
                                1900 :EXCHANGE TWO OR MORE COMMANDS IN THE SEQUENCE
```

Each program step contains all of the information that we controlled earlier (glide on and off, up-scale or down, staccato or legato, and one of 8 tempos) and when the program is run, each step will be taken in turn and an arpeggiation of the keys held down performed using the status of the parameters specified by that step. At the end of the program it jumps back to its beginning and the sequence of arpeggiations repeats.

Each step of the program is "written" in exactly the same way that we set the parameters earlier; in fact, as you'll soon realize, you were in effect writing the first step then. The key to forming these steps into programs is the PNTR : BACK/ADV block of pads on the command keyboard. The pointer (PNTR) refers to the program step that you're writing.

One quick example should get the idea across. We'll write a program that sweeps up the keyboard at a moderate tempo, re-articulating each note, followed by a quick legato run down-scale. Program the first step by touching these keys - TEMPO:4, DIR:UP, GLID:OFF, ART:STAC. That takes care of the up part.

Now for the down part, begin by touching PNTR:ADV so the commands that we enter next are "pointed" at the second program step (which is step #1 as shown in the displays, the first step is #0) and touch TEMPO:2, DIR:DOWN, GLID:OFF, ART:STAC. Now hold down a big chord structure to hear the full effect of this dual arpeggiation.

Editing an existing program is simply a matter of pointing to the program step that you want to change and entering the changed parameter. To change the first step (#0) in the example above to a slower tempo, for example, touch PNTR:BACK so the display shows 00 and then touch TEMPO:7 (or whatever).

OG93 can handle programs up to 8 arpeggiations deep and, when you begin stacking that many steps, it's easy to get lost. The EDIT:EXCH key helps here by allowing us to remove the step pointed to from the program and replacing it with an instruction for repeat. By backspacing the pointer to step #1 and touching the EDIT:EXCH pad, we cut the program to just the first step, EDIT:EXCH again and the original program step is back in place, so that the entire program runs again. By stepping through the program and causing it to repeat after the 2nd, 3rd, etc. steps, it's fairly easy to locate where in the program a specific sound is coming from and then make changes there.

As you may surmise from the name, the EDIT:EXCH key causes the program step pointed to to be exchanged with a memory buffer location which is initialized to contain the interpreter's repeat code (00). This implies that this key can also be used to exchange two program steps by pointing first to one and touching EDIT:EXCH and then to the next and again EDIT:EXCH. In fact, this is the case; with one exception. The first step of the program may not be the repeat code 00. If it is, the interpreter will lock up as it reads the first step, finds that it's a repeat, so it reads the first step, and so on. OG93 protects against this by checking

```
                1910  :
110A- A4 71     1920        LDY *TEMP     :GET THE COMMAND IN THE TEMPORARY BUFFER
110C- D0 04     1930        BNE ELP0      :IS THE COMMAND FROM TEMP A 0? NO, BRANCH
110E- E0 07     1940        CPX 07        :POINTING TO FIRST COMMAND?
1110- F0 2B     1950        BEQ RTN       :YES, BRANCH. DON'T WRITE ZERO AS FIRST COMMAND
1112- 94 77     1960  ELP0  STY *CSEQ,X   :PUT COMMAND IN THE SEQUENCE SLOT POINTED TO
1114- 85 71     1970        STA *TEMP     :AND THEN SAVE OLD COMMAND IN THE TEMP LOCATION
1116- 60        1980        RTS           :THEN RETURN
                1990  :
                2000  :THE KEY IS "10", CLEAR THE COMMAND SEQUENCE. NOTE THAT THE FIRST
                2010  :ENTRY IN THE SEQUENCE IS NOT CHANGED.
                2020  :
1117- A2 07     2030  CLR   LDX 07        :SET POINT TO FIRST SEQUENCE ENTRY
1119- 86 75     2040        STX *PPNT     :AND SAVE IT
111B- CA        2050        DEX           :DECREMENT THE POINTER(SKIP FIRST ENTRY)
111C- A9 00     2060        LDA 00        :AND GET READY
111E- 8D 20 18  2070        STA DISP      :ZERO THE DISPLAYED EDITOR POINTER
1121- 85 71     2080        STA *TEMP     :AND THE EXCHANGE REGISTER
1123- 95 77     2090  CLLP  STA *CSEQ,X   :ZERO THE SEQUENCE ENTRY
1125- CA        2100        DEX           :AND POINT TO NEXT ENTRY
1126- 10 FB     2110        BPL CLLP      :SOME LEFT, LOOP
1128- 60        2120        RTS           :RETURN
                2130  :
                2140  :NOW WE TEST FOR "E" OR "F", BACKSPACE OR ADVANCE THE EDITOR'S
                2150  :EDITOR'S POINTER TO THE COMMAND SEQUENCE. NOTE THAT INCREMENTING THE
                2160  :POINTER PRODUCES A BACKSPACE.
                2170  :
1129- C0 0E     2180  CNXT  CPY 0E        :IS KEY "E" OR "F"?
112B- 90 18     2190        BCC STMP      :NEITHER AND LESS THAN "E", BRANCH FOR NEXT TEST
112D- F0 0F     2200        BEQ BACK      :IT'S "E", BRANCH TO BACKSPACE
112F- CA        2210        DEX           :IT'S "F", ADVANCE THE POINTER
1130- 30 08     2220        BMI RTN       :AND IF OUT OF RANGE, BRANCH TO LEAVE IMMEDIATELY
1132- 86 75     2230  COUT  STX *PPNT     :SAVE NEW POINTER
                2240  :
                2250  :IN THIS SECTION THE POINTER (WHICH IS 07 FOR THE START OF THE SEQUENCE
                2260  :AND 00 AT THE END) IS CONVERTED TO AN INCREASING NUMBER FROM 0-7 FOR
                2270  :DISPLAY PURPOSES.
                2280  :
1134- 8A        2290        TXA           :POINTER TO THE ACCUM. FOR A CALCULATION
1135- 38        2300        SEC           :PREPARE FOR A SUBTRACTION
1136- E9 08     2310        SBC 08        :TWO'SD COMPLEMENT
1138- 49 FF     2320        EOR 0FF       :COMPLEMENT OF THAT
113A- 8D 20 18  2330        STA DISP      :SHOW VALUE IN THE DISPLAYS
113D- 60        2340  RTN   RTS           :RETURN
                2350  :
                2360  :BACKSPACE POINTER AND MAKE SURE IT IS STILL IN RANGE, THEN BRANCH
                2370  :
113E- E8        2380  BACK  INX           :BACKSPACE THE POINTER
113F- E0 08     2390        CPX 08        :OUT OF RANGE?
1141- F0 FA     2400        BEQ RTN       :YES, BRANCH TO LEAVE IMMEDIATELY
1143- D0 ED     2410        BNE COUT      :NO, BRANCH ALWAYS TO SAVE POINTER, ETC.
                2420  :
                2430  :IF THE KEY IS ONE OF THE TEMPOS, ADD 1 (0 TEMPO NOT ALLOWED) AND
                2440  :FIT IT INTO THE CONTROL SEQUENCE ENTRY POINTED TO
                2450  :
1145- C0 08     2460  STMP  CPY 08        :TEMPO KEY?
1147- B0 0A     2470        BCS SGLD      :NO, BRANCH
1149- C8        2480        INY           :YES, ADD 1 TO KEY #
114A- 29 F0     2490        AND 0F0       :MASK PRESENT TEMPO IN COMMAND TO ZERO
114C- 95 77     2500        STA *CSEQ,X   :SAVE CONTROL FLAGS IN CSEQ TEMPORARILY
114E- 98        2510        TYA           :BRING NEW TEMPO TO ACC
114F- 15 77     2520        ORA *CSEQ,X   :COMBINE WITH OLD CONTROL FLAGS
1151- D0 1A     2530        BNE SAVA      :BRANCH ALWAYS
                2540  :
                2550  :NOW A SERIES OF TESTS WHICH RESULT IN THE CARRY BIT BEING SET OR
                2560  :CLEAR. A SERIES OF ROTATES BRINGS THE CARRY TO THE APPROPRIATE BIT
                2570  :IN THE COMMAND WORD
                2580  :
1153- 2A        2590  SGLD  ROL           :ROTATE THE GLIDE COMMAND BIT TO CARRY
1154- 08        2600        PHP           :AND SAVE THE CARRY ON THE STACK
1155- C0 09     2610        CPY 09        :IS KEY GLIDE ON OR OFF?
1157- F0 12     2620        BEQ ROT1      :9-GLIDE ON, BRANCH
1159- 90 10     2630        BCC ROT1      :8-GLIDE OFF, BRANCH
                2640  :
                2650  :THE KEY WAS NEITHER GLIDE ON NOR OFF, TEST FOR DIRECTION UP OR DOWN
                2660  :
115B- 28        2670  SMOD  PLP           :GET THE OLD GLIDE BIT FROM THE STACK
115C- 2A        2680        ROL           :ROTATE DIRECTION BIT TO CARRY
115D- 08        2690        PHP           :SAVE IT ON STACK
115E- C0 0B     2700        CPY 0B        :IS KEY UP OR DOWN?
1160- F0 08     2710        BEQ ROT2      :B-UP, BRANCH
1162- 90 06     2720        BCC ROT2      :A-DOWN, BRANCH
                2730  :
                2740  :THE KEY HAS TO BE C OR D (STACCATO OR LEGATO)
```

John Simonton designs unusual electronic things and writes about them. He' s one of the most fortunate people you could ever meet. He does the things he loves - and gets paid for it.