Don Lancaster
Synergetics

Cheap Video for Your Heathkit H8

Here's a first look at part of Don Lancaster's latest Sams book, Son of Cheap Video. *The TV 6-5/8 he talks about is a full graphics update of the original TVT-6L we ran in the June 1977 issue of* Kilobaud, *is a PAIA kit and is detailed in Lancaster's* Cheap Video Cookbook.

You'll find things more challenging when you add cheap video to an 8080 or Z-80 system, compared to the easy 6500 conversions. There are several new hassles involved that will get in your way and somehow have to be resolved.

In most cases, these hassles will take extra coding, more low-cost ICs and very careful attention to your system timing. The bottom line is this: Cheap video *should* go on most any 8080 or Z-80 system, but it will take more effort, more code and more parts to get comparable results.

Let's see just what these hassles are. First, we'll look at an 8080 in general to see what the hardware and software differences will be. Then we'll check into a general-use 8080/Z-80 adapter that goes between your computer and the TVT 6-5/8. Finally, we'll show you the software you will need to put cheap video on a Heathkit H8.

We'll assume your system is bus oriented and that your cheap video system is to be a piggyback add-on to an existing RAM plug-in card. We'll further assume the usual 2 MHz 8080 speed. Your RAM should be fast enough that it does not use the READY command to hold up CPU time. We'll also assume your system is big enough that nonvolatile scan firmware is more important than minimizing the total words of scan coding.

Be forewarned that what we are going to look at has only been tested on the Benton Harbor 50-pin bus. While there is no obvious reason why you can't do the same thing on an S-100 bus system or with a Z-80, we have not tried it just yet, and neither should you . . . unless you have a good triggered scope on hand and thoroughly understand the 8080 CPU timing.

Our main 8080 hassles are these:

1. The address bus has garbage on it at times.
2. The program counter usually can change only once every *two* microseconds. This is only half as fast as we need for a reasonable number of characters or chunks on a line.
3. Clocking and timing signals are different.
4. Literal translation of scan programs will be far too slow.

In general, we'll get around hassle #1 by latching and holding both address and upstream tap data lines using suitably spaced timing. We'll beat #2 by adding a "speed doubling" circuit that creates the *illusion* of a once-per-microsecond program counter advance. This illusion will appear only at the display memory and then only

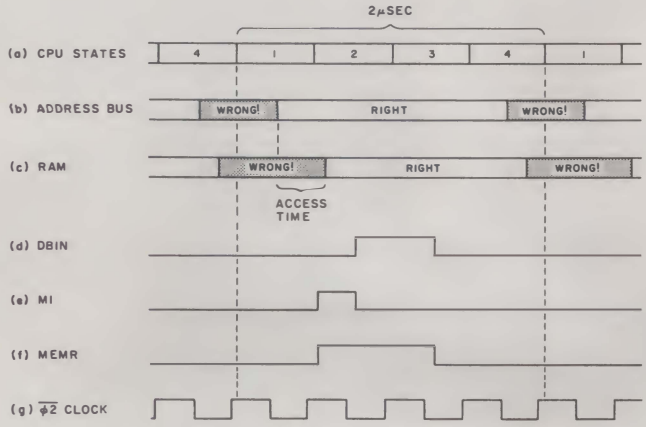


Fig. 1. The H8 is a typical 2 MHz 8080A system. Here are the waveforms involved in reading a NOP command out of RAM.

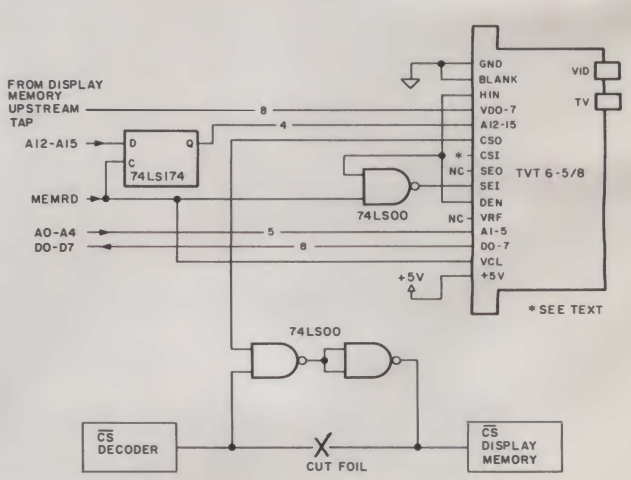
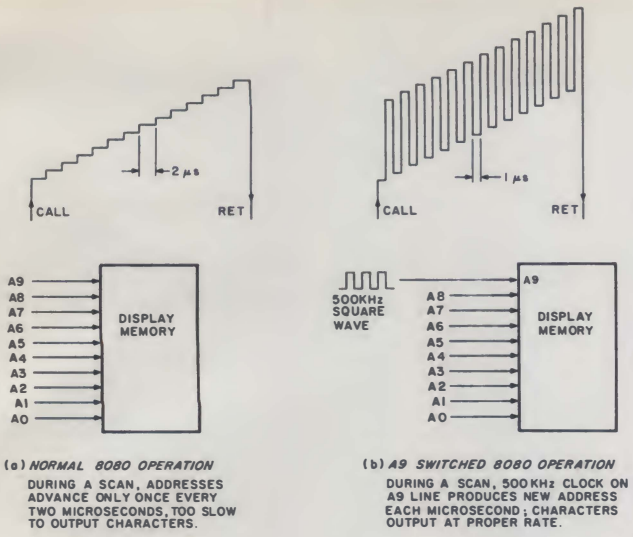


Fig. 2. Minimum 8080A-TVT 6-5/8 interface is limited to 2 usec character or chunk times.



(a) *NORMAL 8080 OPERATION*
DURING A SCAN, ADDRESSES ADVANCE ONLY ONCE EVERY TWO MICROSECONDS, TOO SLOW TO OUTPUT CHARACTERS.

(b) *A9 SWITCHED 8080 OPERATION*
DURING A SCAN, 500 KHz CLOCK ON A9 LINE PRODUCES NEW ADDRESS EACH MICROSECOND; CHARACTERS OUTPUT AT PROPER RATE.

Fig. 3. A stock 8080 system can't change display memory addresses each microsecond. Here's how to use A9 switching for speedup.

during a TVT scan. Everything else stays the usual speed. Hassle #3 goes away when we solve #2. Finally, we can get scan software that is fast enough by using the powerful register-to-register commands of the 8080 or by using brute force (all ROM, non-modifying) coding.

On to the fine print.

Hardware

Suppose we have a normal, functional H8 executing a string of no operations (NOP) from a plug-in RAM card. What will this timing look like? How can we trick the H8 into using the same sort of timing—with add-ons—to run a TVT 6-5/8? Fig. 1 gives us some clues.

Execution of a NOP takes two microseconds (actually, slightly less than this on the H8). Four CPU states (Fig. 1a), each taking around half a microsecond, are involved. The object of these four states is to put the program counter on the address bus, read an addressed memory location, enter it into the CPU and then act on the command. When the CPU finds out the command is a NOP, it will spend the tail end of the cycle essentially doing nothing.

Our first hassle appears in Fig. 1b. We see that the address bus only has the correct information on it three-quarters of the time. For the remaining quarter of the time, the address bus has invalid information on it. Now, if we address a memory with the wrong address, we will, of course, get the wrong information out of the memory. Worse still, since the memory has its own access time to contend with, the amount of time that useful information comes out of the memory is even shorter than the time the address bus is valid (Fig. 1c). So, the bad news is that both data and address have all kinds of holes in them and don't seem directly usable.

There are some system-level signals that may help us out of this bind. Signal DBIN in Fig. 1d determines the time when the CPU *must* have valid data; but this signal is not available on the system bus . . . for a very good reason. Anyone who tries to use this signal will be cutting into the CPU's *own* processing time and degrading performance. Instead, two signals are derived for bus use. These signals occur early enough so that enables, decoding, settling times and so on are complete before the CPU needs valid data. These signals are called M1 (Fig. 1e) and MEMR (Fig. 1f).

M1 starts after the address is valid but ends before DBIN. MEMR includes both the M1 and DBIN times. *Unfortunately, both M1 and MEMR start before we are sure that the memory is outputting valid data.* The theory here is that output enables and bus access can take place during the *same* time that the memory is still accessing itself, so long as everything ends up stable by the start of DBIN time. A final waveform we will find useful is the $\overline{\phi 2}$ system clock shown in Fig. 1g.

The least we can get away with and still get cheap video on an 8080 is latching the upper four address lines. If we don't do this, all the commands out of our TVT instruction decoder PROM, including the row commands and the sync pulses, will have big holes chopped in them.

Fig. 2 shows a minimum 8080-to-TVT 6-5/8 interface. In this circuit, +5, ground, blanking, the upstream tap and the data bus are connected in the usual way. Address lines A12 through A15 are connected to a latch that catches the valid addresses. This is done on the *leading edge* of the memory read command, MEMR.

Our chip select output CSO is shown going to an AND gate that gives us an external negative logic OR combination of the old display memory chip select and the one needed for TVT scanning. A foil cut is involved here. The chip select input, CSI, is shown permanently enabled. Depending on your decode PROM, this can go to a TVT enable switch, do nothing or be used as an internal chip select combiner, eliminating the external gate.

The TVT is only allowed to gain data bus control during a scan and then only when the computer wants to read it. To do this, we use the computer's memory read MEMR command and NAND it with the decode enable, DEN, to get a suitable scan enable SEI input.

MEMR also goes to the clock input of the TVT 6-5/8. But, since our load command in the TVT is derived from the *falling* edge of VCL, it is the *trailing* edge of MEMR that loads our video shift register. The time difference of about 750 nanoseconds gives our character generator more than enough time to produce a valid output.

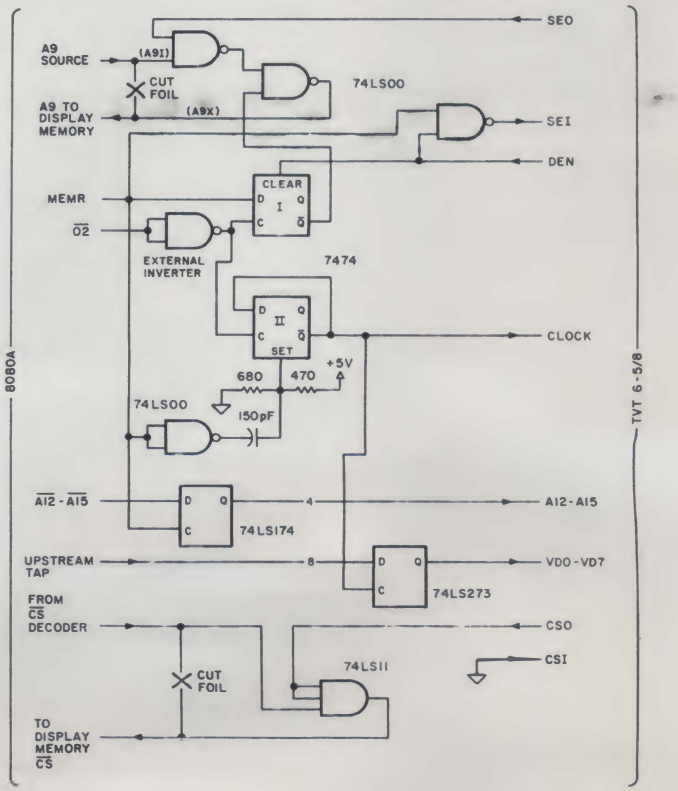


Fig. 4. Speed-doubling 8080A-TVT 6-5/8 interface gives 1 usec character or chunk times.

Now, this is a quick and dirty circuit that you may want to try just to get *some* video out of your 8080 in a hurry. But, there are several problems we still have to attack to get something good enough for final system use.

One minor hang-up is that you may only have complements of your data bus or address bus available. We'll soon see how to change the coding in your Scan and Decode PROMs to get around this. The coding, of course, has to be changed anyway since the 8080 gets all bent out of shape when it receives 6502 commands. Inverters or inverting gates can also be used to invert bus, clock, data or control lines as needed.

The big hassle is that the character or chunk times will be *two* microseconds each, rather than just one. This means that, so far, even a 32 character line won't run at normal horizontal scan frequencies. Beating this particular hassle soundly about the head and ears is the key to practical cheap video on the 8080.

But how?

Speed Doubling Via A9 Switching

We want to get our chunk and character times down to a decent rate of one microsecond. We can either speed up the microprocessor or else do something else that creates the *illusion* of a microprocessor speedup at the display memory and in the adapter circuits.

Speedup may be easy for you if you have a Z-80, provided your display memory is also fast enough to not use the READY command. If you do run faster, you probably would like to latch the upstream tap data to make sure you have enough processing time for your character generator. While a simple speedup will work in some systems, there is a much better way called *A9 switching*.

The object of A9 switching is to create the *illusion* of a once-per-microsecond address advance at the display memory. Fig. 3 gives us details on how this works. We break our most significant display space address line and connect it to a carefully timed 500 kHz square wave during a scan. For a 16 × 64 or a 12 × 80 alphanumeric display, this will be address line A9.

Now, a 500 kHz square wave is low for one microsecond and high for another one. While all the regular addresses *below* A9 are changing at their usual two-microsecond rate, A9 is busy addressing one character or chunk location on the first microsecond and another location on the second. Thus, we get characters or chunks out of our display memory at a one-per-microsecond speed.

But why on earth use A9? Wouldn't it be simpler to use A0 instead? If we do this, we would have to add an address multiplexer to all inputs of the display memory—a 10-pole double throw switch or its Tri-state equivalent. This is obviously something we want to avoid if we are piggybacking video onto an existing memory card. All A9 switching takes a single foil cut and some add-on wires to the memory card.

There is a catch. It is a "yeah-but" rather than a "gotcha." *The characters and chunks are no longer in the display memory in sequential order if you use A9 switching.* So, your cursor or controlling loader software has to have a few words added to complement A9 each successive location.

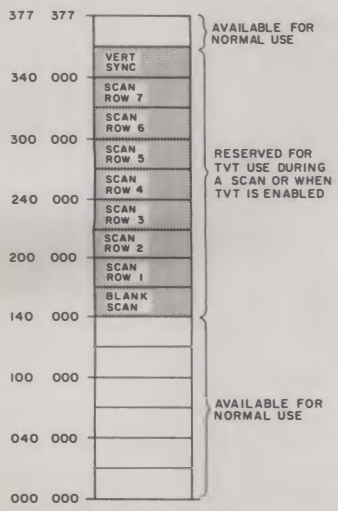


Fig. 6. H8 address map.

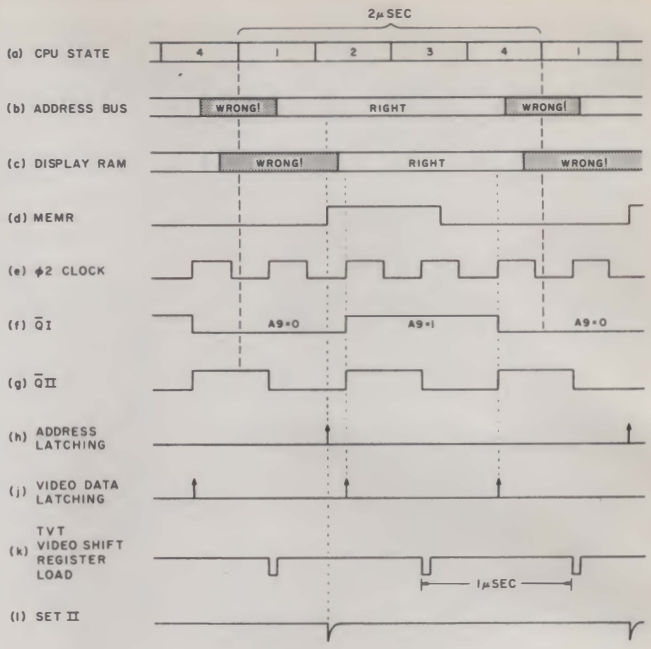


Fig. 5. Speed-doubling waveforms.

For instance, say your display memory starts at 000 000. The next character or chunk will be at 002 000. Your characters will follow in this order:

1st character	000 000
2nd character	002 000
3rd character	000 001
4th character	002 001
5th character	000 002
6th character	002 002
.	.
.	.
.	.
1022nd character	003 376
1023rd character	001 377
1024th character	003 377

This seems awful, but it works. And it is a simple way to double the apparent memory access speed of an 8080 so we can get information out of RAM once per microsecond under block access. And all it takes to do the job is some simple hardware between computer and TVT, a few software words and one extra foil cut on the memory. The hardware involved is shown in Fig. 4, along with the timing details of Fig. 5.

Two new D-flip-flops are added to our interface. The first delays and expands the MEMR signal to give us a controlled phase 500 kHz square wave we can use for the speed doubling A9 address switching. The second divides the system clock by two and is used to latch the video data and to provide a TVT clock.

Waveforms (a), (b), (c) and (d) in Fig. 5 are as before. Waveform (e) is a Ø2 clock, which has to be an inverted replica of the Heath bus $\overline{Ø2}$ clock signal. Waveform (f) shows us the 500 kHz square wave that results when we clock MEMR. Since the clocking is delayed from the MEMR leading edge, the flip-flop's output is wider than MEMR and is almost a microsecond long. This results in a square wave that is low for one microsecond and high for the next, locked to (but following) MEMR.

This particular flip-flop is only allowed to run *during* a scan. Otherwise, it is held high by DEN. The uppermost two gates combine the old A9 information with the speed-doubling new A9 signal, acting as a single pole, double throw selector switch. During computer times, the display memory A9 line is connected to the computer. During scan microinstruction times, the display memory A9 line is connected so it is low for one microsecond and high for the next.

Waveform (g) shows us the one megahertz clock we get by dividing down Ø2. This clock is used to sample and latch the display memory output immediately after the data is valid and then latch again one microsecond later, well after the A9

658-HD8
PROM NUMBER

□ = "0" ■ = "1"
(POSITIVE LOGIC)

USE FOR TVT 6-5/8 ON AN 8080 SYSTEM WITH INVERTED $\overline{A12}$, $\overline{A13}$, $\overline{A14}$, $\overline{A15}$ LINES.

CG LINE "2" IS USED AS GRAPHICS "BLANKING" OUTPUT.

CG LINE "4" IS USED AS GRAPHICS "UPPER-LOWER" CHUNK SELECT OUTPUT.

		INPUTS		OUTPUTS							
	WORD #	WHAT DOES THIS WORD DO?	HEX OP-CODE	Q8 CS OUT	Q7 SCAN ENABLE	Q6 DECODE ENABLE	Q5 VERTICAL SYNC	Q4 (SPARE)	Q3 CG LINE "4"	Q2 CG LINE "2"	Q1 CG LINE "1"
TVT ENABLED	0	NORMAL	C∅	■	■	□	□	□	□	□	□
	1	VERTICAL SYNC	d∅	■	■	□	■	□	□	□	□
	2	LINE 7 SCAN	27	□	□	■	□	□	■	■	■
	3	LINE 6 SCAN	26	□	□	■	□	□	■	■	□
	4	LINE 5 SCAN	25	□	□	■	□	□	■	□	■
	5	LINE 4 SCAN	24	□	□	■	□	□	■	□	□
	6	LINE 3 SCAN	23	□	□	■	□	□	□	■	■
	7	LINE 2 SCAN	22	□	□	■	□	□	□	■	□
	8	LINE 1 SCAN	21	□	□	■	□	□	□	□	■
	9	BLANK SCAN	20	□	□	■	□	□	□	□	□
	10	NORMAL	C0	■	■	□	□	□	□	□	□
	11	NORMAL	C0	■	■	□	□	□	□	□	□
	12	NORMAL	C0	■	■	□	□	□	□	□	□
	13	NORMAL	C0	■	■	□	□	□	□	□	□
	14	NORMAL	C0	■	■	□	□	□	□	□	□
	15	NORMAL	C0	■	■	□	□	□	□	□	□
TVT DISABLED	16	NORMAL	C0	■	■	□	□	□	□	□	□
	17	NORMAL	C0	■	■	□	□	□	□	□	□
	18	NORMAL	C0	■	■	□	□	□	□	□	□
	19	NORMAL	C0	■	■	□	□	□	□	□	□
	20	NORMAL	C0	■	■	□	□	□	□	□	□
	21	NORMAL	C0	■	■	□	□	□	□	□	□
	22	NORMAL	C0	■	■	□	□	□	□	□	□
	23	NORMAL	C0	■	■	□	□	□	□	□	□
	24	NORMAL	C0	■	■	□	□	□	□	□	□
	25	NORMAL	C0	■	■	□	□	□	□	□	□
	26	NORMAL	C0	■	■	□	□	□	□	□	□
	27	NORMAL	C0	■	■	□	□	□	□	□	□
	28	NORMAL	C0	■	■	□	□	□	□	□	□
	29	NORMAL	C0	■	■	□	□	□	□	□	□
	30	NORMAL	C0	■	■	□	□	□	□	□	□
	31	NORMAL	C0	■	■	□	□	□	□	□	□

Fig. 7. Truth table for 8080 Decode PROM having inverted address inputs (used on Heathkit H8).

change has been accepted. The first sample gives us an A9 = 0 data value, while the second handles the A9 = 1 case. The TVT's video shift register is clocked on the falling edge of this one megahertz clock. Since there is a one-half microsecond delay between the leading and trailing clock edges, enough time is available for the character generator or the data-to-video converter to accept the latched video data and process it.

Our A9-generating flip-flop automatically initializes itself on MEMR since it is simply delaying this signal. But the clock-dividing flip-flop can be in either state at the *beginning* of a scan microinstruction. Unless we somehow initialize this flip-flop to the right state, we'll get garbage out of the display memory caused by sampling at the wrong times.

We initialize this clock-dividing flip-flop by inverting MEMR and using the leading edge to SET the divide flip-flop to the desired state. This initialization is very important since the usual CALL instruction preceding the scan microinstruction has an odd number of clock cycles in it.

TVT scan enabling and the display memory chip selecting are done the same way as the slower interface of Fig. 2. We enable the TVT Scan Enable Input (SEI) only during MEMR time to give us data for a scan microinstruction only when it is called for and only when the computer will allow data bus access. The display memory chip select is a negative logic OR of the computer's chip select and the CSO that the TVT provides.

Our speed doubling interface takes two foil cuts on the memory board—one on the A9 address line and one on the chip select line. All other connections are add-ons derived from signals available on a typical plug-in memory card. Five low-cost integrated circuits are involved in this particular adapter.

Software

Let's take a look at the PROM firmware and some of the software involved in getting cheap video on your 8080A system. For right now, we'll stick to the older address-mapped and subroutine-scanned methods we used in the *Cheap Video Cookbook*. Most likely you can simplify things a great deal by going to the Scungy Video* route of break-mapping and interrupt-scanning. The strong Input/Output commands in the 8080A make this a very attractive idea.

If you use address mapping, refer to the computer memory map shown in Fig. 6. A block of addresses from 6K to 60K is reserved for TVT use when the TVT is enabled. On the H8, this leaves the bottom 8K for the PAM monitor and operating system and 16K for enough RAM to hold a display memory and run Extended BASIC at the same time. The uppermost 4K of addresses are also available as needed.

Should you want more address space for other uses, you can use the TVT enable to free addresses during non-display times. You can also go the Scungy Video route and use I/O instructions and a parallel port instead of address mapping the row commands. Yet another alternative is to use further decoding to activate the TVT only during valid display memory addresses. For instance, if you are only using 1K of display memory, 3K of all the scan blocks can be used for other

*Scungy Video is an alternate method and is detailed in Chapter 1 of *Son of Cheap Video*.

658-HS64
PROM NUMBER

□ = "0" ■ = "1"
(POSITIVE LOGIC)

USE FOR TVT 6-5/8 ON AN 8080 SYSTEM WITH TRUE A0-A7 LINES AND INVERTED DATA BUS NO REPACKING

	INPUTS		OUTPUTS							
WORD #	WHAT DOES THIS WORD DO?	HEX OP-CODE	Q8 DB7	Q7 DB6	Q6 DB5	Q5 DB4	Q4 DB3	Q3 DB2	Q2 DB1	Q1 DB0
0	NOP	FF	■	■	■	■	■	■	■	■
1	NOP	FF	■	■	■	■	■	■	■	■
2	NOP	FF	■	■	■	■	■	■	■	■
3	NOP	FF	■	■	■	■	■	■	■	■
4	NOP	FF	■	■	■	■	■	■	■	■
5	NOP	FF	■	■	■	■	■	■	■	■
6	NOP	FF	■	■	■	■	■	■	■	■
7	NOP	FF	■	■	■	■	■	■	■	■
8	NOP	FF	■	■	■	■	■	■	■	■
9	NOP	FF	■	■	■	■	■	■	■	■
10	NOP	FF	■	■	■	■	■	■	■	■
11	NOP	FF	■	■	■	■	■	■	■	■
12	NOP	FF	■	■	■	■	■	■	■	■
13	NOP	FF	■	■	■	■	■	■	■	■
14	NOP	FF	■	■	■	■	■	■	■	■
15	NOP	FF	■	■	■	■	■	■	■	■
16	NOP	FF	■	■	■	■	■	■	■	■
17	NOP	FF	■	■	■	■	■	■	■	■
18	NOP	FF	■	■	■	■	■	■	■	■
19	NOP	FF	■	■	■	■	■	■	■	■
20	NOP	FF	■	■	■	■	■	■	■	■
21	NOP	FF	■	■	■	■	■	■	■	■
22	NOP	FF	■	■	■	■	■	■	■	■
23	NOP	FF	■	■	■	■	■	■	■	■
24	NOP	FF	■	■	■	■	■	■	■	■
25	NOP	FF	■	■	■	■	■	■	■	■
26	NOP	FF	■	■	■	■	■	■	■	■
27	NOP	FF	■	■	■	■	■	■	■	■
28	NOP	FF	■	■	■	■	■	■	■	■
29	NOP	FF	■	■	■	■	■	■	■	■
30	NOP	FF	■	■	■	■	■	■	■	■
31	RET	36	□	□	■	■	□	■	■	□

Fig. 8. Truth table for 8080 Scan PROM having no repacking, true address inputs and inverted data outputs.

658-HS80
PROM NUMBER

□ = "0" ■ = "1"
(POSITIVE LOGIC)

USE ONLY FOR 80 CHARACTER REPACKED LINES ON AN 8080 SYSTEM WITH TRUE A0-A7 LINES AND INVERTED DATA BUS.

INPUTS			OUTPUTS								
WORD #	WHAT DOES THIS WORD DO?	HEX OP-CODE	Q8 DB7	Q7 DB6	Q6 DB5	Q5 DB4	Q4 DB3	Q3 DB2	Q2 DB1	Q1 DB0
0	NOP	FF	■	■	■	■	■	■	■	■
1	NOP	FF	■	■	■	■	■	■	■	■
2	NOP	FF	■	■	■	■	■	■	■	■
3	NOP	FF	■	■	■	■	■	■	■	■
4	NOP	FF	■	■	■	■	■	■	■	■
5	NOP	FF	■	■	■	■	■	■	■	■
6	NOP	FF	■	■	■	■	■	■	■	■
7	NOP	FF	■	■	■	■	■	■	■	■
8	NOP	FF	■	■	■	■	■	■	■	■
9	NOP	FF	■	■	■	■	■	■	■	■
10	NOP	FF	■	■	■	■	■	■	■	■
11	RET	36	□	□	■	■	□	■	■	□
12	NOP	FF	■	■	■	■	■	■	■	■
13	NOP	FF	■	■	■	■	■	■	■	■
14	NOP	FF	■	■	■	■	■	■	■	■
15	NOP	FF	■	■	■	■	■	■	■	■
16	NOP	FF	■	■	■	■	■	■	■	■
17	NOP	FF	■	■	■	■	■	■	■	■
18	NOP	FF	■	■	■	■	■	■	■	■
19	NOP	FF	■	■	■	■	■	■	■	■
20	NOP	FF	■	■	■	■	■	■	■	■
21	RET	36	□	□	■	■	□	■	■	□
22	NOP	FF	■	■	■	■	■	■	■	■
23	NOP	FF	■	■	■	■	■	■	■	■
24	NOP	FF	■	■	■	■	■	■	■	■
25	NOP	FF	■	■	■	■	■	■	■	■
26	NOP	FF	■	■	■	■	■	■	■	■
27	NOP	FF	■	■	■	■	■	■	■	■
28	NOP	FF	■	■	■	■	■	■	■	■
29	NOP	FF	■	■	■	■	■	■	■	■
30	NOP	FF	■	■	■	■	■	■	■	■
31	RET	36	□	□	■	■	□	■	■	□

Fig. 9. Truth table for 80 character 8080 Scan PROM (true address inputs, inverted data outputs).

purposes if you add suitable decoding.

A quick look at the H8-3 memory board shows that only some of the address and data lines are available in their true form; most of them are Inverted. The data-out buffer on this memory card must be disabled for the upstream tap needed by cheap video. This means that the output of our Scan Microinstruction PROM has to directly drive the system data bus and thus must output inverted (negative logic) data. We also see that address lines A13, A14 and A15 aren't available except as complements. The simplest way out of this situation is to code our *Decode* PROM to respond directly to complemented addresses.

Fig. 7 shows us the H8 Decode PROM truth table, 658-HD8. We input lines $\overline{A12}$, $\overline{A13}$, $\overline{A14}$ and $\overline{A15}$, along with a TVT enable using the old CSI line. This PROM outputs code to the row commands of the character generator or else routes blanking and selection commands to a graphics data-to-video converter. The Decode PROM also outputs system controlling signals DEN, SEO, CSO and the vertical sync VRF pulses.

Since we are using complemented address inputs, this PROM runs "backwards" from the earlier PROMs we looked at. The net result of a "frontwards" PROM with true address inputs or a "backwards" PROM with inverted address inputs is the same.

Holding the CSI line positive disables the TVT and frees most all addresses for other uses. Grounding CSI enables the TVT scanning and reserves the needed address blocks for TVT use. This particular PROM coding needs an external AND gate for chip selection and combination.

There are two types of *Scan* PROM coding we might like to use, depending on whether we are using "binary" line lengths or are repacking "non-binary" line lengths for maximum memory efficiency. Fig. 8 shows a Scan PROM coding intended for 64 character lines, but usable for 32 character lines, most graphics and other lengths *without* memory repacking. This is numbered 658-HS64.

We use a NOP to advance the program counter in the computer and an RET coding to return from the called scan microinstruction. Since we are outputting complemented data, these outputs are inverted. On the H8, address lines A0 through A6 are available in true form, so we do not have to complement the address Inputs. Thus, our Scan PROMs run "frontwards" but output complemented code.

We can use the 658-HS80 Scan PROM truth table in Fig. 9 for memory repacked scans of 80 characters per line, three lines per page. Once again, this PROM coding is driven by true addresses and outputs complementary data directly to the H8 $\overline{\text{data}}$ bus.

Our address lines are connected differently on an 8080 system than on a 6502. Remember that we used every *second* address change on the 6502 to advance our Scan PROM *one* count. On an 8080 we use *every* address change to advance the Scan PROM *one* count, but use A9 switching to get two characters out of memory per one Scan PROM count advance. Either way, the Scan PROM responds to an input address change once every *two* microseconds, and everything comes out even.

This means that, in general on an 8080 system, the Scan PROM's inputs are usually connected to one address line *less* than usual for a 6502 system. Fig. 10 shows our address line management for an 8080 adapter. It also shows how two new switches can be added along with a gate to let you use either a 658-HS64 or a 658-HS80 Scan PROM on an 8080 system without needing any rewiring.

Several examples will show how this address management works.

1. For 32 character lines using speed doubling, use PROM 658-HS64 and set your switches to A4 = "+," A5 = "+" and "32."
2. For 64 character lines using speed doubling, use PROM 658-HS64 and set your switches to A4 = "A4," A5 = "+" and "32."
3. For 80 character lines using speed doubling and memory repacking, use PROM 658-HS80 and set your switches to A4 = "A4," A5 = "A5" and "64."

In our first example, the upper half of a Scan PROM is cycled through in 16 counts lasting 32 microseconds. In the second example, the entire Scan PROM is cycled through in 32 counts lasting 64 microseconds. In the final example, if we wanted to, the entire Scan PROM could be scanned in 32 counts lasting 256 microseconds. But with memory repacking and A9 switching, we only use slightly under a third of the 80 line Scan PROM *per scan*, ending up with ten counts per scan lasting 80 microseconds.

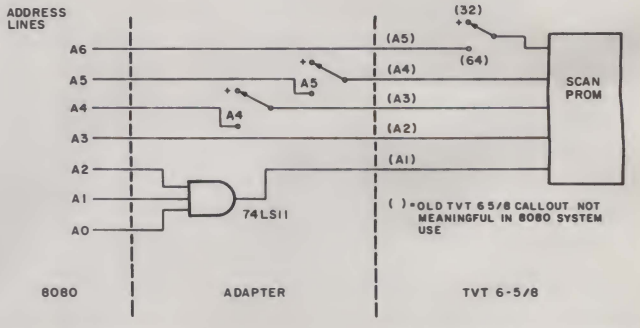


Fig. 10. The Scan PROM address inputs on the TVT 6-5/8 have to be redefined for 8080 use. The gate and switches let you run ordinary or repacked memory PROMs without wire changes.

Your turn: Show the Scan PROM truth table and switch settings for an H8 Scan of 40 repacked characters per line.

Front Panel Interaction

The H8 front panel works by interrupting a running program once every two milliseconds. If we try to run scan software and the front panel at the same time, the display will be badly torn up. So, we can either turn the front panel off during display times or else combine the front panel and the video scan into a single program. Just turning the front panel off is far simpler and usually all you will need to do.

The H8 front panel monitor does have a "turn the display off" software word. But this won't help us. While this command shortens the interrupt and keeps it from lighting the display, the interrupt still exists.

One hardware solution is shown in Fig. 11. A new switch is added to the front panel that prevents timer-generated level 10 interrupts from happening. This, in turn, keeps the panel display off and the video display in one piece. This switch will be very handy during your initial test and debugging of video displays. You should only turn *off* the front panel after you have a video display, and turn it back *on* before returning to other uses. The RST/0 command does bypass this switch so that you can reset under any conditions.

This switch will most likely *not* be needed when your properly designed and debugged scan software is operational. You probably can eliminate it from the final use circuitry.

The obvious question is how to use software instead. We have a good old DI, or "disable interrupts," command in the 8080 instruction set. Can't we simply use this?

Unfortunately, there is one very noisy gotcha that may keep you from doing this—unless you are careful.

If you try an immediate DI command in an H8 program, the speaker will latch on and *stay* on. That little beep you get when you hit the GO key—or any other key—needs *two* more interrupts *after* your program starts. No interrupts, no stopping. The two interrupts time out a four millisecond tic for the horn circuit.

The H8 front panel monitor needs a few milliseconds *after* it is exited before you can disable any interrupts. If you disable an interrupt too soon you will lock the speaker on.

You can use the DI command to turn off the front panel, but you must delay at least five milliseconds *after* your program starts or the speaker won't quit. Thus, one properly placed software word is all you need to get full front panel and video display compatibility.

Test Software

Two useful test routines are shown in Fig. 12. Fig. 12a checks Scan PROM access and operation. If this test fails, you are either incorrectly picking up scan microinstructions or are missing them entirely. Erratic switching between 311 (return) and 000 (no operation) means you have speed-doubling problems. All 000s means you are never activating the Scan PROM, while all 311s means you are permanently trying to return from a Scan Microinstruction call. This particular test works with either HS64 or HS80 Scan PROMs and can have the address switches in any position.

Your turn: Why?

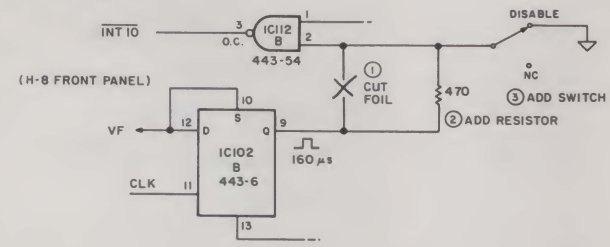
Don't ever try going beyond this test if the test fails. If you cannot read the proper return from a scan microinstruction, it will not execute, and anything else you add in the way of software or time or effort will only compound the felony.

Test sequence Fig. 12b lets you transfer control of the H8 from computer to TVT scanning and back again. Note that the test coding differs for each Scan PROM and that each Scan PROM has to have the address switches set as shown.

The scanning process is adjusted to output a TV horizontal scan at normal scan frequencies. In a completely working system with a disabled front panel, you'll get a continuous series of vertical stripes. This corresponds to *the seventh dot* row of a random character load. A wildly wrong horizontal scan frequency usually means the wrong switch settings or the wrong Scan PROM. Vertical stripes that have teeth in them may be caused by erratic data latching or improper speed-doubling operation.

While these two tests appear trivially simple, don't overlook them as major debugging aids. If these two won't go, no other software will run either.

(a) SCHEMATIC



(b) PICTORIAL

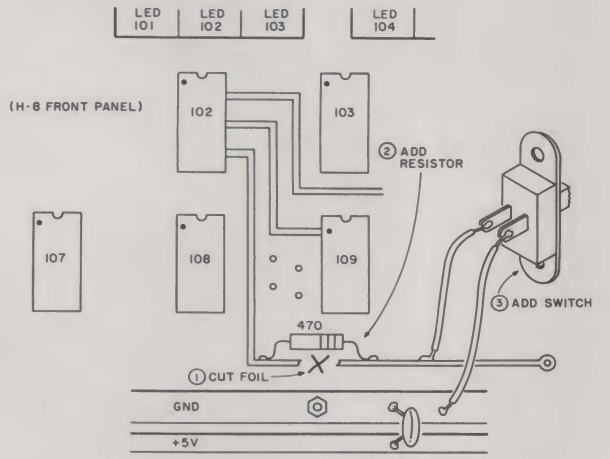


Fig. 11. A switch to temporarily defeat the H8 front panel display will be useful for TVT debugging and checkout.

A. To verify that the Scan Microinstruction is alive and well:

read

300 376	for	000	(NOP)
300 377	for	311	(RET)
301 000	for	000	(NOP)

Either the HS64 or the HS80 Scan PROM may be used.
The address switches may be in any position.

B. To pass control to and from the Scan Microinstruction at a TV Horizontal rate:

For Scan PROM HS64

Set switches to "32"; A5 = " + " and A4 = "A4"

START →	040 100	CALL	315 010 320	Scan seventh dot row
↑	040 103	JMP	303 100 040	Repeat

For Scan PROM HS80

Set switches to "64"; A5 = "A5" and A4 = "A4"

START →	040 100	CALL	315 030 320	Scan seventh dot row
↑	040 103	JMP	303 100 040	Repeat

This will display continuous vertical stripes that correspond to the *seventh* dot row of a random character load. The front panel should be switch disabled during viewing times.

H8 Scan time is 63 microseconds for a horizontal scan frequency of 15.898 kHz. There is no vertical sync.

Fig. 12. Two test routines useful in 8080/TVT debugging.

Self-Modifying vs Brute Force Scans

The obvious next thing to do is take the old 6502 scan software programs and literally translate them, replacing a CALL for a JSR and so on. But we really get into trouble in a hurry if we try this. First, some commands will be longer or shorter than their 6502 counterparts, messing up the critical horizontal-edge-to-horizontal-edge timing. Worse yet, the execution time of an 8080 working with literally translated 6502 commands is pitifully slow—so slow that the critical timing loop may take over 30 microseconds, compared to the 21 used in the 6502. This makes the long horizontal lines so long we don't want to even think about using them.

One solution is to make the 8080 into an 8080 rather than an imitation 6502. You can do this using the fast register-to-register transfer commands and get your loop times down only slightly longer than those in the 6502 programs.

But is this really what we want in an 8080 system? Remember that on a bare-bones KIM-1 our back was to the wall in finding room for a scan program. We *had* to get by with the absolute minimum-length scan programs in order to get any video at all.

One result of this restriction was that our scan code was *self-modifying*. This meant that the scan program computed its next set of memory locations rather than looking them up. This, in turn, meant that the scan program *had* to be in RAM during final operation, at least on a KIM.

Usually our 8080 systems have enough RAM and PROM available that we needn't worry too much about minimizing code. So, why not use *brute force* coding that calls each scan address as it is needed? We can store the whole scan program in ROM or PROM this way and never have to load it again . . . or worry about it bombing when something bad happens in RAM.

Brute force coding will also be much faster. It will be much easier to write, modify and debug. But, as usual, there is a price. Brute force coding can be much longer than self-modifying coding. On a one-line display, this turns out to be a no-hassle 43 words versus the 30 words we needed on a KIM with self-modifying code. But on a long and involved program such as a 24 × 80 double-stuffed scan, it could take 600 or more words of code to get us by. Still, that's only little over half a 2708 or slightly over a quarter of a 2716 EPROM and no real big deal these days.

Let's use this brute force approach to generate a simple one line display and then apply it to a 12 × 80 scan program.

1 × 56 Scan Program

Fig. 13 shows a brute force scan program for a one line, 56 character no-interlace 8080/TVT 6-5/8 display. Each successive dot row is called by a scan subroutine as it is needed. We start in 040 100 with a short blank scan to get us off on the right foot. Then we sequentially call dot rows 1 through 7 of the characters to be displayed. This live scanning is followed by a vertical sync pulse.

After this, a word that sets the number of blank scans is loaded in the accumulator (365). As many blank scans as needed are generated in turn. Each time a blank scan is completed, the accumulator word is decremented till the word hits zero. At that time, the program jumps to the top line blank scan and repeats for the next field.

Unlike a 6502, an 8080 can take an even or an odd number of *half* microseconds to complete an instruction. In most scan programs, some equalization will be needed to make up for this half-microsecond jitter. The command MOVAA, or "move the accumulator to itself," takes 2.5 microseconds and is a benign instruction. This lets us shift timing by half a microsecond if used once and by one microsecond if used

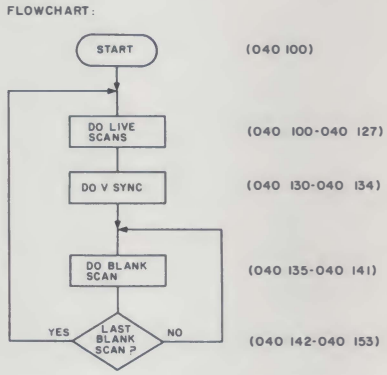


Fig. 13a. Program flowchart.

uP-8080A Start-JMP 040 100 Displayed 340 004 to 340 037
System-H8 End-RST/0 342 004 to 342 037
Program Space 040 100 to 040 152
(43 words)

```
START-->040 100   CALL  315  017  140   Do short blank scan
        040 103   CALL  315  004  160   Scan Dot row #1
        040 106   CALL  315  004  200   Scan Dot row #2
        040 111   CALL  315  004  220   Scan Dot row #3

        040 114   CALL  315  004  240   Scan Dot row #4
        040 117   CALL  315  004  260   Scan Dot row #5
        040 122   CALL  315  004  300   Scan Dot row #6
        040 125   CALL  315  004  320   Scan Dot row #7

        040 130   LDA   072  000  340   Output Vertical sync pulse
        040 133   MVIA  076  365        Load # of blank scans
        040 135-->CALL  315  011  140   Do blank scan
        040 140   DCRA  075             One less scan

        040 141   MOVAA 177             Equalize 2.5 microseconds
        040 142   JNZ   302 (135)(040)  One more blank scan?
        040 145   MOVAA 177             Equalize 5.0 microseconds
        040 146   MOVAA 177                 continued

        040 147   DI    363             Shut off horn
        040 150   JMP   303 (100)(040)  Go to live scans
```

Mods:

To relocate display space, use program jumpers on memory card or else change starting address of dot scans.
To put both halves of display space closer together, use A4 switching rather than A9 switching.
For double height characters, repeat scan of each dot row twice.

Notes:

- TVT 6-5/8 must be connected via an 8080 adapter, and both the 658-HD8 and 658-HS64 PROMs must be in circuit for the program to run.
- Horizontal frequency 15.174 kHz; Vertical frequency 59.976 Hz. 2500 second hum bar.
- Address switches must be in "32", A5 = " + ", and A4 = "A4" positions.
- Character sequence goes 340 004; 342 004; 340 005; 342 005; 340 006; 342 006; 340 007. . . . . .
- ( ) denotes an absolute address that is program location sensitive.
- This program is not self-modifying and may be placed in PROM or ROM.

Fig. 13 Program for a one line, 56-character, no-interlace TVT 6-5/8 8080 raster scan.

Fig. 14. Program for a 12 line, 80-character-per-line, full-interlace, double-stuffed TVT 6-5/8 raster scan.

```
                    uP-8080A          Start-RUN 040 100
                    System-H8         End-RST/0
                        Displayed     340 010 to 343 377
                    Program Space     040 100 to 042 007 (455 words)

                              (even field)
START--->040 100   CALL  315  023  140     Do short blank scan

         040 103   CALL  315  010  140     Scan dot row 0, character line 1
         040 106   CALL  315  010  200           "      2       "          1
         040 111   CALL  315  010  240           "      4       "          1
         040 114   CALL  315  010  300           "      6       "          1
         040 117   CALL  315  010  140     Do blank scan

         040 122   CALL  315  060  140     Scan dot row 0, character line 2
         040 125   CALL  315  060  200           "      2       "          2
         040 130   CALL  315  060  240           "      4       "          2
         040 133   CALL  315  060  300           "      6       "          2
         040 136   CALL  315  060  140     Do blank scan

         040 141   CALL  315  130  140     Scan dot row 0, character line 3
         040 144   CALL  315  130  200           "      2       "          3
         040 147   CALL  315  130  240           "      4       "          3
         040 152   CALL  315  130  300           "      6       "          3
         040 155   CALL  315  130  140     Do blank scan

         040 160   CALL  315  210  140     Scan dot row 0, character line 4
         040 163   CALL  315  210  200           "      2       "          4
         040 166   CALL  315  210  240           "      4       "          4
         040 171   CALL  315  210  300           "      6       "          4
         040 174   CALL  315  210  140     Do blank scan

         040 177   CALL  315  260  140     Scan dot row 0, character line 5
         040 202   CALL  315  260  200           "      2       "          5
         040 205   CALL  315  260  240           "      4       "          5
         040 210   CALL  315  260  300           "      6       "          5
         040 213   CALL  315  260  140     Do blank scan

         040 216   CALL  315  330  140     Scan dot row 0, character line 6
         040 221   CALL  315  330  200           "      2       "          6
         040 224   CALL  315  330  240           "      4       "          6
         040 227   CALL  315  330  300           "      6       "          6
         040 232   CALL  315  330  140     Do blank scan

         040 235   CALL  315  010  141     Scan dot row 0, character line 7
         040 240   CALL  315  010  201           "      2       "          7
         040 243   CALL  315  010  241           "      4       "          7
         040 246   CALL  315  010  301           "      6       "          7
         040 251   CALL  315  010  141     Do blank scan
         040 254   CALL  315  060  141     Scan dot row 0, character line 8
         040 257   CALL  315  060  201           "      2       "          8
         040 262   CALL  315  060  241           "      4       "          8
         040 265   CALL  315  060  301           "      6       "          8
         040 270   CALL  315  060  141     Do blank scan

         040 273   CALL  315  130  141     Scan dot row 0, character line 9
         040 276   CALL  315  130  201           "      2       "          9
         040 301   CALL  315  130  241           "      4       "          9
         040 304   CALL  315  130  301           "      6       "          9
         040 307   CALL  315  130  141     Do blank scan

         040 312   CALL  315  210  141     Scan dot row 0, character line 10
         040 315   CALL  315  210  201           "      2       "          10
         040 320   CALL  315  210  241           "      4       "          10
         040 323   CALL  315  210  301           "      6       "          10
         040 326   CALL  315  210  141     Do blank scan

         040 331   CALL  315  260  141     Scan dot row 0, character line 11
         040 334   CALL  315  260  201           "      2       "          11
         040 337   CALL  315  260  241           "      4       "          11
         040 342   CALL  315  260  301           "      6       "          11
         040 345   CALL  315  260  141     Do blank scan

         040 350   CALL  315  330  141     Scan dot row 0, character line 12
         040 353   CALL  315  330  201           "      2       "          12
         040 356   CALL  315  330  241           "      4       "          12
         040 361   CALL  315  330  301           "      6       "          12
         040 364   CALL  315  330  141     Do blank scan

         040 367   MVIA  076  006          Delay 48.5 microseconds
         040 371   DCRA  075                     continued
         040 372   JNZ   302 (371)(040)          continued

         040 375   LDA   072  000  340     Output //VERTICAL SYNC// pulse
         041 000   CALL  315  363  140     Do short blank scan
         041 003   LDA   072  000  000     Delay 6.5 microseconds
         041 006   MVIA  076  175          Load # of vertical blank scans

      ->041 010   CALL  315  015  140     Do //BLANK VERTICAL SCANS//
      |  041 013   DCRA  075               One less blank scan
      |  041 014   MOVAA 177               Equalize 2.5 microseconds
      |_ 041 015   JNZ   302 (010)(041)    Repeat blank scans if not done

         041 020   MOVAA 177               Equalize 5 microseconds
         041 021   MOVAA 177                     continued
         041 022   DI    363               Shut off horn
```

twice. This is the purpose of those strange "177" instructions in the program.

In step 040 147, we disable the interrupts. This turns off our front panel but does so late enough that we will not lock the speaker on. Since the code is not self-modifying, you can put it in your choice of RAM, ROM, PROM, EPROM or E²PROM. Naturally, you'll want to check things out in RAM first before committing yourself to permanent code.

Your turn: Show the coding needed for 1×32, 1×64 and 1×80 scans.

As a hint that will save you lots of trial and error or calculations, keep your blank initial scan *nine* counts short of the live scans and keep the retrace blank scans *five* counts short of your live scans. A stationary or near-stationary hum bar is picked up by adjusting *040 134* as needed. A more obvious route to shorter scans is to simply use the 1×56 and load blanks as needed in unused character locations.

TV Retrace Hassles

Calling and returning from a subroutine takes around 13.5 microseconds on a typical 8080. Two of these microseconds are spent on the live scan, leaving us with a retrace time of 11.5 microseconds. Since the H8 is slightly faster than this, our available retrace time is around 11.2 microseconds.

Naturally, we would like to keep our retrace times as short as possible. This lets you put more characters on the line for standard horizontal rates or lets you run long character lines with more nearly normal horizontal frequencies.

But 11 microseconds may not be enough time for your monitor or TV set to cleanly get from the end of one line to the beginning of the next. For most monitors and some TV sets, 11 microseconds will be just barely enough.

If you are having trouble displaying all the characters, here are some hints that may help you:

- Your simplest out is to adjust

the display centering so that the first character is always legible. Always stop short of the maximum display length on your statements.

- Use the *maximum* possible width. Raising the width coil inductance (see *Cheap Video Cookbook*, Fig. 3-33) can lengthen the needed retrace time.
- Use a longer-than-needed character line and put permanent blanks where they are called for.
- Add equalization to lengthen each CALL sequence. While this is the obvious and cleanest route, it can add many words to a brute force scan program.
- If you *thoroughly* understand TV horizontal scanning *and* have a decent scope *and* full TV documentation, modify the flyback capacitor value as needed to get a faster retrace. But, be careful to not exceed the peak allowable horizontal output transistor voltage when you do this.

More Characters

Our 1 × 56 scan has several obvious limitations. From this starting point, we'll want to add interlace, double stuffing and lots more characters.

The optimum number of characters or chunks per line seems to be 56 for an H8 system using A9 switching for speed doubling. This 56-character length lets you use a standard horizontal frequency. You can display on either a color or a black and white set.

But there seems to be something magical about 80 character lines that appeals to people, even though this many characters are hard to read and are rarely, if ever, needed. So, to prove it can be done, we're going to show you how to display 80 character lines on your H8 and then put those lines on a TV with unmodified video bandwidth or over an rf modulator. Remember, though, that we'll have to run at a reduced horizontal rate, which will take width and hold modifications to your small-screen, transformer-operated, *Photofact*-available, black and white set. Furthermore, your wrong choice

```
041 023   JMP   303 (100)(041)    Jump to odd field

(041 026 to 041 077 are spares)
                    (odd field)
041 100   CALL  315  023  140     Do short blank scan

041 103   CALL  315  010  160     Scan dot row 1, character line 1
041 106   CALL  315  010  220           "      3        "         1
041 111   CALL  315  010  260           "      5        "         1
041 114   CALL  315  010  320           "      7        "         1
041 117   CALL  315  010  140     Do blank scan

041 122   CALL  315  060  160     Scan dot row 1, character line 2
041 125   CALL  315  060  220           "      3        "         2
041 130   CALL  315  060  260           "      5        "         2
041 133   CALL  315  060  320           "      7        "         2
041 136   CALL  315  060  140     Do blank scan

041 141   CALL  315  130  160     Scan dot row 1, character line 3
041 144   CALL  315  130  220           "      3        "         3
041 147   CALL  315  130  260           "      5        "         3
041 152   CALL  315  130  320           "      7        "         3
041 155   CALL  315  130  140     Do blank scan

041 160   CALL  315  210  160     Scan dot row 1, character line 4
041 163   CALL  315  210  220           "      3        "         4
041 166   CALL  315  210  260           "      5        "         4
041 171   CALL  315  210  320           "      7        "         4
041 174   CALL  315  210  140     Do blank scan

041 177   CALL  315  260  160     Scan dot row 1, character line 5
041 202   CALL  315  260  220           "      3        "         5
041 205   CALL  315  260  260           "      5        "         5
041 210   CALL  315  260  320           "      7        "         5
041 213   CALL  315  260  140     Do blank scan

041 216   CALL  315  330  160     Scan dot row 1, character line 6
041 221   CALL  315  330  220           "      3        "         6
041 224   CALL  315  330  260           "      5        "         6
041 227   CALL  315  330  320           "      7        "         6
041 232   CALL  315  330  140     Do blank scan

041 235   CALL  315  010  161     Scan dot row 1, character line 7
041 240   CALL  315  010  221           "      3        "         7
041 243   CALL  315  010  261           "      5        "         7
041 246   CALL  315  010  321           "      7        "         7
041 251   CALL  315  010  141     Do blank scan

041 254   CALL  315  060  161     Scan dot row 1, character line 8
041 257   CALL  315  060  221           "      3        "         8
041 262   CALL  315  060  261           "      5        "         8
041 265   CALL  315  060  321           "      7        "         8
041 270   CALL  315  060  141     Do blank scan
041 273   CALL  315  130  161     Scan dot row 1, character line 9
041 276   CALL  315  130  221           "      3        "         9
041 301   CALL  315  130  261           "      5        "         9
041 304   CALL  315  130  321           "      7        "         9
041 307   CALL  315  130  141     Do blank scan

041 312   CALL  315  210  161     Scan dot row 1, character line 10
041 315   CALL  315  210  221           "      3        "         10
041 320   CALL  315  210  261           "      5        "         10
041 323   CALL  315  210  321           "      7        "         10
041 326   CALL  315  210  141     Do blank scan

041 331   CALL  315  260  161     Scan dot row 1, character line 11
041 334   CALL  315  260  221           "      3        "         11
041 337   CALL  315  260  261           "      5        "         11
041 342   CALL  315  260  321           "      7        "         11
041 345   CALL  315  260  141     Do blank scan

041 350   CALL  315  330  161     Scan dot row 1, character line 12
041 353   CALL  315  330  221           "      3        "         12
041 356   CALL  315  330  261           "      5        "         12
041 361   CALL  315  330  321           "      7        "         12
041 364   CALL  315  330  141     Do blank scan

041 367   LDA   072  000  340     Output //VERTICAL SYNC// pulse
041 372   MVIA  076  175          Load # of vertical blank scans

041 374   CALL  315  015  140     Do  //BLANK VERTICAL SCANS//
041 377   DCRA  075               One less blank scan
042 000   MOVAA 177               Equalize 2.5 microseconds
042 001   JNZ   302 (374)(041)    Repeat blank scans if not done

042 004   MOVAA 177               Equalize 5 microseconds
042 005   MOVAA 177                  continued
042 006   DI    363               Shut off horn
042 007   JMP   303 (100)(040)
```

Notes:

- TVT 6-5/8 must be connected via an 8080 adapter, and both the 658-HD8 and 658-HS80 PROMs must be in circuit for the program to run.
- Address switches must be in "64"; A5 = "A5"; and A4 = "A4" positions.
- Horizontal frequency = 11.191 kHz Vertical frequency = 60.006 Hertz. 166 second hum bar.
- This program is not self-modifying and may be placed in PROM or ROM.
- Character sequences goes 340 000; 350 000; 340 001; 350 001; 340 002; 350 002; 340 003 .....

() denotes an absolute address that is program location sensitive.

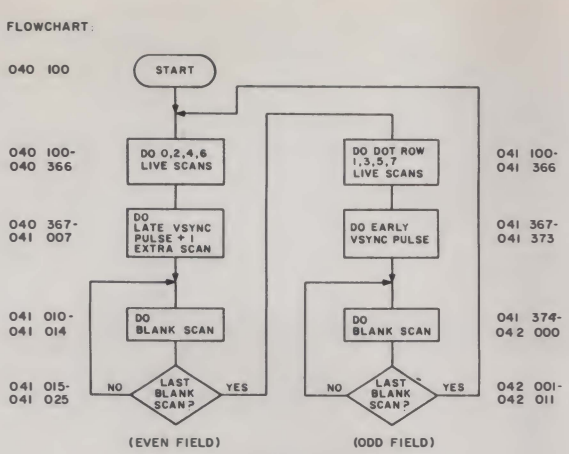


Fig. 14a. Program flowchart.

of set could sing objectionably.

12 Lines of 80 Characters

A brute force, interlaced, double-stuffed 12 × 80 scan program appears in Fig. 14. You can easily modify it for 24 × 80 or even 36 × 80 displays if you like. With the double stuffing, the 12 × 80 display uses slightly less than one-third of the H8 throughput time. By going to suitable transparency techniques, you can save two-thirds of the computer time to transparently run other programs such as Extended BASIC.

We've shown you this scan program with its memory space at 340 010 to 343 377. This assumes you have at least two RAM cards in your H8 and have put this particular one "out on top" with the "56K" jumper on the memory card. You may want to relocate things later, but this is a handy place to start.

The TVT 6-5/8 is attached to the memory card by way of an 8080 adapter similar to Figs. 4 and 10. The TVT does place certain use restrictions on the 340 000 to 360 000 computer address space, since any activity here also gives you a vertical sync pulse that might disrupt an enabled display. You *can* use this space for a display memory RAM; you should *not* use this area for the scan program or the computer stack. If you do use this page for display memory RAM, you will have to watch your cursor program carefully if transparent character entry is important to you.

You'll find the 12 × 80 program shown in two separate fields. We have an *even* field that puts down the even dot rows of all the characters and an *odd* field that puts down the odd dot rows of all the characters. When combined, these fields form an interlaced and double-stuffed frame. Having the two fields separate is handy for debugging. By jumping a field back on itself, you can display all-even or all-odd fields to fix coding errors or make format changes.

The scan program runs just about the same way the earlier 1 × 56 program did. First, there is a short blank scan; then we put down the even dot rows of all the characters. Then we equalize, followed by a *late* vertical sync pulse, at the same time taking up one entire *extra* horizontal scan time. Then we run the usual blank vertical scans, completing the field.

When the field is finished, we jump to the odd field, run a short blank scan and put down all the odd dot rows of all the characters. After this, we run an *early* vertical sync pulse and go on to the usual number of vertical blank scans. The scan sequence repeats by jumping to the start of an even field.

The early and late vertical sync pulses differ by half a horizontal line. When you combine this half a line with the *extra* horizontal line picked up only in the even scan, you end up with an interlaced scan of 373 whole lines taking one 30 Hz frame. This 30 Hz frame consists of two 60 Hz fields of 186.5 lines each.

The 658-HS80 Scan PROM lets you repack the 80 character lines so you can use your display memory space efficiently. Fig. 15 shows how the characters are arranged in RAM. While this looks like a royal mess, a few extra cursor words are all we need to straighten things out. This is often a reasonable trade-off for letting us do long lines with an 8080 in the first place and freeing up 600 or so words of system RAM for other uses.

Your turn: Show the coding for 24 × 80, 32 × 80, 16 × 56, 32 × 56, 16 × 64 and 32 × 64 scan programs. Show ways of significantly shortening the 12 × 80 scan program while staying PROM compatible. Try: (1) using only one vertical blanking sequence and minimizing blank sequences and unused code words; (2) using I/O commands to free address space; (3) using interrupt rather than subroutine mapping.

Note that you'll use the HS64 PROM for 64 and shorter character lines and most graphics, while the HS80 PROM is usually reserved for 80 character lines. You can do 40 character lines with the HS64 without repacking, or else you can use your memory more efficiently by going to a specially coded HS40 PROM that uses repacking. Repacking saves you RAM space but needs a few extra words in the cursor program and takes a special Scan

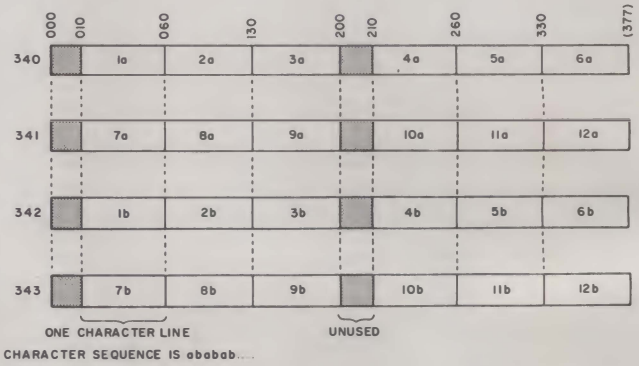


Fig. 15. Display memory map for 12 × 80 scan.

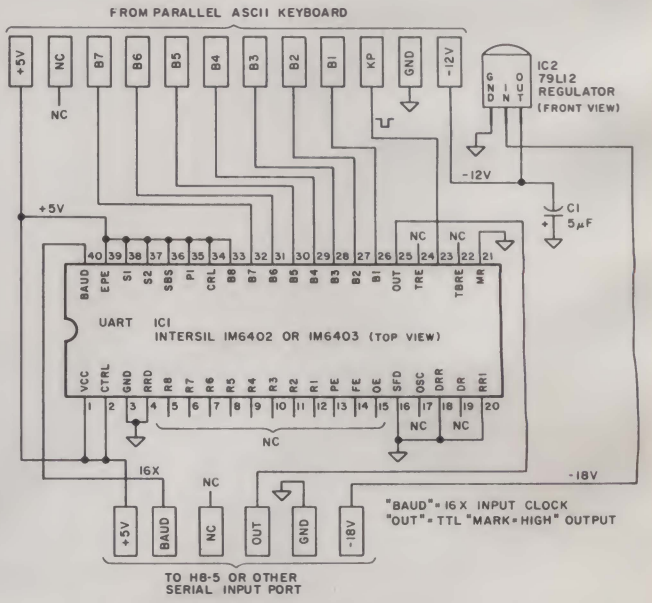


Fig. 16. This keyboard serial adapter lets you connect a keyboard to a serial computer input.

PROM.

A Keyboard Serial Adapter

If you have an H8-2 parallel interface card, it should be fairly easy to interface almost any old ASCII keyboard and encoder. You could do this essentially the same way we did it on the parallel KIM inputs back in the *Cheap Video Cookbook*, but the H8-2 card is an expensive option and you might not already have one on hand. More likely, you'll be using the H8-5 serial interface card instead, since you need this one for the usual cassette and remote terminal uses.

Most ASCII keyboards and encoders provide only a parallel (all-the-bits-at-once) output. To enter a serial port, we have to convert this parallel word into a serial (one-bit-at-a-time) sequence. A simple adapter to do this is shown in Fig. 16.

The circuit can use the transmitter half of nearly any old UART (universal asynchronous receiver-transmitter). We first looked at UARTs back in Chapter 7 of the *TVT Cookbook*. You'll find this circuit easier and more inexpensive when you use a modern, single-supply CMOS chip such as an *Intersil* IM6402 or IM6403.

The keyboard serial adapter works by borrowing power from the H8-5 serial interface and feeding +5 volts and optionally −12 volts to your existing keyboard. Your existing keyboard outputs are most likely available in *parallel* or "all-at-once" form. These parallel outputs and a normally-high keypressed *strobe* are routed to the transmitter side of the UART in the adapter. This UART also borrows a 16X baud clock from the H8-5.

When you press a key, a serial output is generated by the UART. This serial output is then routed to your computer's serial interface and received just as if it came from a terminal.

You may need as many as five leads between your adapter and the H8-5. One is ground, two are for power, one is the 16X baud rate clock that goes *to* the adapter and the final is the serial output that comes *from* the adapter. Fig. 17 shows you how to connect, both pictorially and schematically, your adapter to your H8-5. You can either hard-wire these connections or add a new connector of your own.

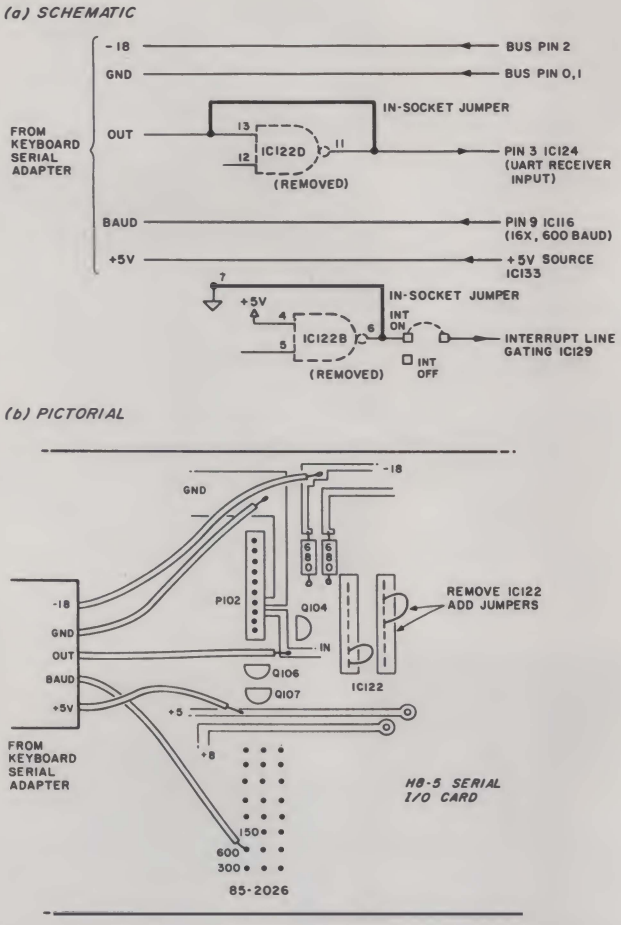


Fig. 17. Connecting your keyboard serial adapter to an H8-5 interface.

On your H8-5 board, integrated circuit IC122 is removed and replaced with two jumpers inserted in the socket as shown. The pin-11-to-pin-13 jumper gives you direct access to the serial input on the UART present inside the H8-5. The pin-6-to-pin-7 jumper lets you use the keyboard in a *polled* mode. This polled operation gives you a transparent scan program and frees the interrupts for other uses.

The H8 has to be software-programmed to use your new adapter. A simple test sequence that will enter the last-pressed key into the accumulator and display it for you is shown in Fig. 18.

The H8-5 is first initialized with a *mode* instruction. You can use 312 and output it to port 373. This picks two stop bits, ignores parity, uses a seven-bit word and runs with a 16X clock. Next, you continue to initialize the H8-5 by giving a *command* instruction to the same port. This time, use 004 and once again output it to port 373. This command instruction will enable only the receiver in the H8-5 interface.

After the mode instruction and the command instruction are routed to the interface, you are free to read characters. You do this by inputting from port 372. The final loop in the test program does this continuously.

As you press a key, its ASCII value will appear in the left three digits of the "AF" Register display. For instance, a lowercase "b" will read 142, while an uppercase "B" will read 102.

There are a few gotchas in this simple test program, so you'll want to improve it for actual use as part of a cursor. Note that this simple program continuously *reread's* characters instead of reading each one just once. To beat this, there is available a "character ready" (R×RDY) flag that is set when

```
                    uP-8080A            Start-JMP 040 100
                    System-H8+H8/5      End-RST/0
                    Program Space       040 100 to 040 113 (13 words)

START ->040 100   MVIA  076  312        Initialize mode instruction
        040 102   OUT   323  373             continued
        040 104   MVIA  076  004        Initialize command instruction
        040 106   OUT   323  373             continued

      ->040 110   IN    333  372        Read Keyboard
        040 112   JMP   303 (110)(040)  Loop
```

Notes:

- This test program displays a pressed key received via the Keyboard Serial Adapter. To run the program, use: RST/0-REG-PC-ALTER-0-4-0-1-0-0-ALTER-REG-AF-GO.
- ASCII characters should appear as the three leftmost digits on the display. For instance, "A" = 101, "a" = 141, "6" = 066, and "CR" = 015.
- ( ) Denotes an absolute address that is relocation sensitive.

Fig. 18. Keyboard serial adapter test program.

the character first arrives and is reset as soon as the computer uses the character for the first time.

To use a character only once, input from port 373, AND what you get with 002 and test the result. A nonzero result means you have a new character ready to enter. A zero result says you have already used the character on-hand and should ignore it. We'll see an example on this shortly.

The UART doing the transmitting (in the adapter) and the one doing the receiving (in the H8-5) must agree on the baud rate and the baud clock factor. Usually, the H8-5 will be set on 600 baud and 16X clocks with internal jumpers. If not, or if you are on a different system, be sure that the transmitting UART and the receiving UART are on speaking terms with each other.

Note that your initialization of the *mode* and *command* words should be done only once after reset and before any input/output activity. If you don't initialize, you'll get no characters at all, and if you continuously re-initialize, characters will get dumped before you can use them.

Your keyboard serial adapter is very flexible. For instance, go over the data sheets to find a whole unused UART receiver on the low number pins. The −12 volt supply is an option. You can eliminate it if you already have −12 on hand or use a keyboard that doesn't need it. You can also use the old-style UARTs that need −12 by removing the connections on pin #2 and jumpering to −12.

Should you use the IM6403, you can eliminate the 16X baud rate line by connecting a 3.58 MHz color TV crystal between pins 17 and 40 while grounding pin 3 of the IM6403. This will output characters for you at 110 baud. Your computer's serial input will also have to be jumpered or programmed to use this new data rate.

As shown, the keyboard serial adapter is programmed to provide a permanent one in the transmitted ASCII bit #8, is continuously enabled, has no parity, uses two stop bits and has an eight-bit word length. You can change any or all of these by reprogramming the hard-foil connections of pins 33 through 39 of the UART. Our circuit assumes the keyboard outputs positive logic and uses a narrow goes-to-ground-from-positive-high strobe that is low only when data is valid. The output is a simple TTL logic level. There is no need to convert to RS-232 or Teletype current loops for a short interface connection.

Your turn: Show how to use your keyboard serial adapter with only two wires between computer and keyboard, including all power supply connections. Hint: Use the IM6403 with a crystal and a CMOS-encoded keyboard. Change the current when you want to send a zero and sense this current at the computer end.

If you really want to get fancy, use ultrasonic or infrared transducers to give you zero connections between keyboard and computer. This will, of course, take batteries inside the keyboard, or will it?

8080 Cursor Software

Many of the ideas we have already used for our previous cur-

Fig. 19. Program for a one-line, 56-character TVT 6-5/8 8080 raster scan integrated minimum cursor.

```
uP-8080A      Start-JMP 040 100        Displayed      340 004 to 340 037
System-H8     End-RST/0                               342 004 to 342 037
                                       Program Space  040 100 to 040 341
                                       Registers Used -B,     H,     L

Main scan sequence:

START ->040 100   MVIA  076  312        Initialize MODE for H8-3
        040 102   OUT   323  373              continued
        040 104   MVIA  076  004        Initialize COMMAND for H8-3
        040 106   OUT   323  373              continued

      ->040 110   CALL  315  017  140   Do short BLANK SCAN
        040 113   CALL  315  004  160   Scan Dot row #1
        040 116   CALL  315  004  200   Scan Dot row #2
        040 121   CALL  315  004  220   Scan Dot row #3

        040 124   CALL  315  004  240   Scan Dot row #4
        040 127   CALL  315  004  260   Scan Dot row #5
        040 132   CALL  315  004  300   Scan Dot row #6
        040 135   CALL  315  004  320   Scan Dot row #7

        040 140   MVIB  006  364        Load number of blank scans in B
        040 142   IN    333  373        Is a new key pressed?
        040 144   ANI   346  002        Mask kepressed bit
        040 146   JZ    312 (154)(040)  No, continue scan
040 220 040 151   CALL  315 (220)(040)  Yes, go to cursor
      ->040 154   CALL  315  015  140   Do equalizing BLANK SCAN
        040 157   LDA   072  000  340   Output vertical sync pulse
        040 162   MOVBA 170  170        Get number of blank scans back

      ->040 164   CALL  315  011  140   Do BLANK SCAN
        040 167   DCRA  075             One less scan
        040 170   MOVAA 177             Equalize 2.5 microseconds
        040 171   JNZ   302 (164)(040)  Do another blank scan?
        040 174   MOVAA 177  177        Equalize 5 microseconds

        040 176   DI    363             Shut Off Horn
        040 177   JMP   303 (110)(040)  Go to new field

CURSOR
RETURN

Cursor Processing Subroutine:

040 151 040 220   MOVAH 174             Get upper cursor address
(Enter) 040 221   ANI   346  375        Mask A9 out
        040 223   CPI   376  340        Is upper page address valid?
        040 225   JZ    312 (233)(040)  Yes, OK to continue

040 260 040 230   CALL  315 (260)(040)  No, clear screen via subroutine
      ->040 233   MOVAL 175             Get lower cursor address
        040 234   ANI   346  037        Put it on the screen
        040 236   MOVLA 157             Replace lower cursor

        040 237   IN    333  372        Get character
        040 241   CPI   376  015        Is it Carriage Return (Erase)?
        040 243   JZ    312 (260)(040)  Yes, clear screen via subroutine
040 300 040 246   CALL  315 (300)(040)  No, enter character via subroutine

        040 251   RET   311             Return to scan program

          (040 251 through 040 257 are spares; not used)

040 320 040 260   CALL  315 (320)(040)  go to clear screen subroutine
        040 263   MVIB  006  331        Equalize # of blank scans remaining
040 154 040 265   RET   311             Return to Processing
(Exit)
```

Enter Character and Increment Subroutine:

```
enter ──► 040 300   MOVMA 167              Store character at cursed location
          040 301   MOVAH 174              Get upper cursor word
          040 302   XRI   356  002         Change address A9
          040 304   MOVHA 147              Replace upper cursor word

          040 305   ANI   346  002         Is address A9 now zero?
exit 1 ──►040 307   RNZ   300              No, return
          040 310   INXH  043              Yes, increment HL (cursor address)
exit 2 ──►040 311   RET   311              Return to Processing

Enter ──► 040 320   LXIH  041 (004)(340)   Home Cursor
          040 323   MVIA  076  040         Load Space
          040 325   CALL  315 (300)(040)   Enter space via ECI subroutine
          040 330   MVIA  076  040         Is it the end of the screen?

          040 332   CMPL  275                   continued...
          040 333   JNC   302 (323)(040)   No, add more spaces
          040 336   LXIH  041 (004)(340)   Yes, home cursor
Exit ──►  040 341   RET   311              Return to Processing
```

Notes:

- TVT6-5/8 must be connected via an 8080 adapter and both the 658-HD8 and 658-HS64 PROMs must be in circuit for the program to run. Character entry via keyboard, a keyboard serial adapter and the H8-3 serial interface card.
- All characters and all control commands are entered on the screen, except for carriage return (CR), which clears the screen.
- Horizontal frequency is 15.174 kHz; Vertical frequency is 59.976 Hz. 2500 second hum bar.
- Address switches must be in "32"; A5 = " + "; and A4 = "A4" positions.
- Character sequence goes 340 004; 342 004; 340 005; 342 005; 340 006; 342 006; 340 007. . . .
- This program is not self-modifying and may be placed in PROM or ROM. Register "B" is used for temporary storage; Registers "HL" are used to hold the cursor address.
- To aid in debugging, replace 040 147 with 000 and manually defeat front panel interrupt. To shorten number of characters displayed for a tv with limited width, use 040 337 value of 005 or higher.
- ( ) denotes an absolute address that is program location sensitive.

sors will carry over to 8080 cursor design. One new hassle we'll pick up is the straightening-out process needed to undo the A9 speed doubling. But this is more than offset by the easier and simpler code using all the available 8080 registers, particularly the 16-bit wide HL register that is ideal for cursor location storage.

Let's look at a simple cursor that ties the keyboard input to an 8080 display. We'll use the 1 × 56 display to keep things simple. The program and a flowchart are shown in Fig. 19.

For convenience, we've left this program in several pieces, omitted a visible cursor and done only "good enough" equalization. While you can use this program for a one-line point-of-sale terminal, as a deaf communicator or in a prompting environment, chances are that you'll want to pick up these bits and pieces and then combine them with the best of the earlier cursors to do your own thing.

Our main scan sequence is about the same as the old 1 × 56 scan program of Fig. 13. We've added some words at the start that initialize our H8-5 serial interface so it will accept a keyboard input by way of the keyboard serial adapter. Our brute force scans are called for next as needed to give us a line of characters.

After the characters are down, we test to see if a new key has been pressed. If not, we output a vertical sync pulse, run the blank vertical retrace scans, and then jump up and repeat everything for the next field. Note that we do *not* re-initialize the serial interface each time. We simply loop back to the start of the next field.

Now, if a key has been pressed, we jump to the new *Cursor Processing* subroutine at 040 220 through 040 251. This cursor processing subroutine first checks to make sure the HL register is holding a valid cursor location. If it isn't, the screen is erased and the cursor fixed before anything happens to other programs in the machine.

We then get a character and test it to see if it is a CR, or carriage return. If it is a CR, we erase the screen and home the cursor. CR was chosen over CAN in this example as it seems more appropriate for a one-line display. You can, of course, use any decoding you like.

If any key *but* the carriage return is pressed, the character is entered. This is done by way of an enter-character-and-increment, or ECI, subroutine. This ECI subroutine is fancier than the ones we used before, since we have the A9 switching to contend with. Some new rules and a few extra code words take care of this for us.

Remember that the A9 switching was used to let us get characters out of the 8080 fast enough to be useful. To do this, the display characters are out of order. Specifically, for our 1 × 56 display, the character sequence goes like this:

```
1st character ..... 340 004
2nd character ..... 342 004
3rd character ..... 340 005
4th character ..... 342 005
      . . . . .
55th character ..... 340 037
56th character ..... 342 037
```

Now every time we enter a character, we want to go on to the next one. So, we first *change* A9. To do this, we use an exclusive OR 002 of the H register. This will automatically make A9 a one for a particular character, a zero for the next character, a one for yet the next character and so on.

If A9 goes from a zero to a one, we need do nothing further. If A9 goes from a one to a zero, however, we need to move onto the next pair of character slots in memory. To do this, we increment the HL register that contains the cursor.

So, we change A9 *every* new character but increment our HL cursor only every *second* character. All the A9 switching mess is magically eliminated with nothing but eight or so program words.

Your turn: Show an all-the-bells-and-whistles cursor for a 24 × 80 display, including a visible cursor, full equalization and transparency, all cursor motions and the usual goodies.

As with the 6502 systems, there is virtually no limit to how fancy your cursor programs can become. All it takes are extra words of machine-language code to do almost anything you can dream up. ■

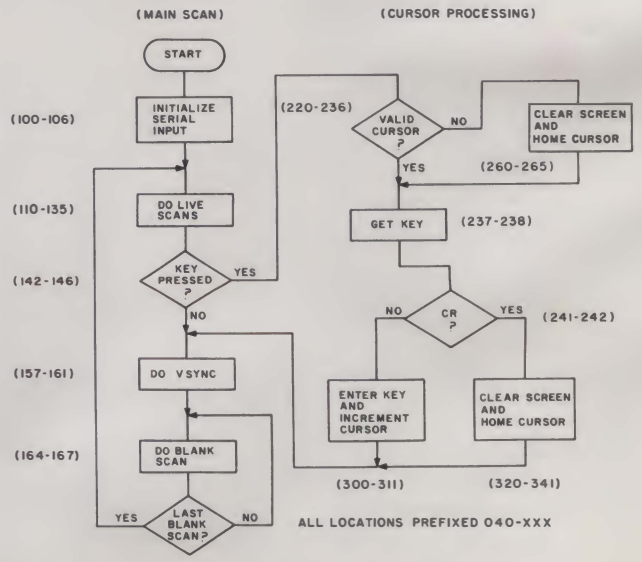


Fig. 19a. Program flowchart.