We've gathered the family to show you why

PERCOM's™ Number 1

in cassette data systems for microcomputers.

Pardon us for doing a little boasting, but we're proud of our family. Proud of each member's reputation for performance and reliability. And pleased that we can offer the best in cassette data systems and data terminal interfacing at low, home-computing prices.

It took more than guts and a little luck to forge a position of leadership. We're number 1 because you get more when you buy PERCOM™. The reason, simply, is experience. Every product described in this ad is based on nearly 10 years of crucial involvement in the design and manufacture of computer peripherals that use cassettes for mass storage.

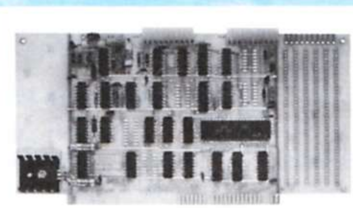
Experience. It's why we developed a more reliable data cassette for home computing. Why our interfacing units provide **both** cassette and data terminal interfacing. Why you get the fastest, most reliable cassette data rates from PERCOM™. Experience. It's *the* reason for PERCOM™.

For your SS-50 bus computer — the CIS-30+

- Interface to data terminal and *two* cassette recorders with a unit only 1/10 the size of SWTP's AC-30.
- Select 30, 60, or 120 bytes per second cassette interfacing, 300, 600 or 1200 baud data terminal interfacing.
- Optional mod kits make CIS-30+ work with *any* microcomputer. (For MITS 680b, ask for Tech Memo TM-CIS-30+—09.)
- KC-Standard/Bi-Phase-M (double frequency) cassette data encoding. Dependable self-clocking operation.
- Ordinary functions may be accomplished with 6800 Mikbug™ monitor.
- Prices: Kit, $79.95; Assembled, $99.95.

Prices include a comprehensive instruction manual. Also available: Test Cassette, Remote Control Kit (for program control of recorders), IC Socket Kit, MITS 680b mod documentation, Universal Adaptor Kit (converts CIS-30+ for use with any computer).

MIKBUG® Motorola, Inc.



For your S-100 computer — the CI-812

- Both cassette and data terminal interfacing on one S-100 bus PC board.
- Interfaces *two* recorders. Record and playback circuits are independent.
- Select 30, 60, 120, or 240 bytes per second cassette interfacing, 110 to 9600 baud data terminal interfacing.
- KC-Standard/Bi-Phase-M (double frequency) encoded cassette data. Dependable self-clocking operation.
- Optional firmware (2708 EPROM) Operating System available.
- Prices: kit, $99.95; assembled, $129.95.

Prices include a comprehensive instruction manual. In addition to the EPROM Operating System, a Test Cassette, Remote Control Kit (for program control of recorders), and an IC Socket Kit are also available.

For your data storage — Pilon-30™ data cassettes

- Orders-of-magnitude improvement in data integrity over ordinary audio cassettes.
- Pilon-coated pressure pad eliminates lint-producing felt pad of standard audio cassettes.
- Smooth pilon coating minimizes erratic tape motion.
- Foam pad spring is energy absorbing. Superior to leaf spring mounted pad which tends to oscillate and cause flutter.
- Five-screw case design virtually precludes deformation during assembly.
- Price: $2.49.

PERCOM™ products may be purchased from home computer dealers nationwide, or may be ordered direct from the factory.*

*Texas residents must include an additional 5% for factory orders. MC & Visa cards honored.

PERCOM™ *'peripherals for personal computing'*

PERCOM

PERCOM DATA COMPANY, INC.
DEPT. K
318 BARNES • GARLAND, TEXAS 75042
Phone: (214) 276-1968

P7

Converting a binary number to one's complement. Write down the binary number. Then *invert* each bit – that is, change each 1 to a 0 and each 0 to a 1. For example, the one's complement of 10110 is 01001.

Converting an octal number to one's complement. Write the octal number. Then *above* each digit put a 7. Now subtract each bottom digit from the top digit.

To convert the octal 0145, for example, you proceed like this:

7	7	7	7
-0	-1	-4	-5
7	6	3	2

Remember that in the binary number that's actually in your computer, each 0 is being inverted into a 1 as you complement. Any *extra* zeros you put in will produce extra ones in the complement. For example, octal 5 is binary 101. But it is also 0101, 00101, 000101, etc., since putting extra zeros in front of a binary number does not change it. But look what happens if you try to get the one's complement (Fig. 1).

An octal 5 can have many different complements; but notice that the only difference between them is the presence of extra ones at the left. The solution is to use only as many ones at the left as will fit the word length of the computer being used. For example, in an eight-bit computer the complement of 5 would be 11111010 binary, or 372 octal.

So, whenever you find the complement of *any* number, always be sure to keep in mind the word length of your computer, and modify the answer to fit your word length. In the case of hobby computers, this problem usually arises on either the Heath H8 computer or any 8008 system, which use octal with an eight-bit word length. Since the leftmost octal digit of any octal number on these computers only stands for two binary digits, the largest it can be is octal 3 (or binary 11). Hence, any complement that starts with a digit *greater* than 3 is wrong. The usual trick is to subtract a 4 from the leftmost digit.

Suppose you want the one's complement of 005. If you follow the rule for converting, you get

777
-005
772

Since the leftmost digit is greater than 3, there is an extra bit. Remove it by subtracting 4 from it, so the actual complement is 372.

Converting a hex number to one's complement. The rule is the same as for octal numbers, except that we write a 15 above each digit and convert hex digits to and from decimal.

The one's complement of hex 68 is hex 97.

15	15
- 6	- 8
9	7

The one's complement of hex 9E is hex 61; we have to convert E to 14:

15	15
- 9	- E (14)
6	1

The one's complement of hex 61 is hex 9E; this time we have to convert 14 to a hex E:

15	15
- 6	- 1
9	14 (E)

The same warnings about extra ones in the complement apply here as when using octal numbers; but we don't usually have to worry about it because in most computer systems the number of bits matches the hex digits exactly. For example, the two hex digits used in eight-bit computers like the 8080 or 6800 match the word length exactly.

Converting numbers in one's complement to two's complement. As mentioned before, most systems use two's rather than one's complements. It's easy to convert from one's to two's complement: add 1. If the one's complement of some number is 110, the two's complement is 111; if it's 61, the two's complement is 62; if it's 9B, the two's complement is 9C – adding 1 to B (which is 11) makes it C (12).

Be careful how you add 1 – it has to be done right. For example, if the one's complement is a binary 101, adding 1 does not give you 102 because a 2 is not allowed in binary! 101 plus 1 is 110 (refer to the table).

Although this is irrelevant anyway since there are other ways of converting, it is of some interest since many microprocessors convert to the two's complement by first finding the one's complement and then adding a 1. For instance, the Intersil 6100 has a CIA (complement and increment accumulator) instruction. (*Increment means to add one.*)

Converting a binary number to its two's complement. Write the binary number. Now find the rightmost 1 and put a vertical line just to the left of it. Invert all bits to the left of this line. Leave the bits to the right of the line unchanged.

Convert the binary number 10110 thus:

1 0 1	1 0
0 1 0	1 0
invert	leave alone

The two's complement of the eight-bit number 00000101 is 11111011:

0 0 0 0 0 1 0	1
1 1 1 1 1 0 1	1
invert	

Converting an octal number to its two's complement. Write the octal number and see whether it has any zeros at its *right* end (ignore zeros in the middle or at the left). If so, put a zero *above* each zero at the right. For instance, if you wanted to convert the octal number 02307500, you would now have

						0	0
0	2	3	0	7	5	0	0

Continue from the right and put an 8 above the next digit and a 7 above each of the others. Finally, subtract each digit from the one above it (see Example 2). The two's complement in this case is 75470300.

7	7	7	7	7	8	0	0
-0	-2	-3	-0	-7	-5	-0	-0
7	5	4	7	0	3	0	0

Example 2.

Just one warning: Everything we said about extra ones in the one's complement conversion applies here, too. For instance, in an eight-bit computer the complement of 005 would be 373, not 773.

If you find this method too hard to remember, you can always convert your octal number to binary, find the two's complement of that, and then convert that back to octal.

Converting a hex number to its two's complement. Look at the hex number to see whether it has any zeros on the right end (ignore zeros in the middle or at the left). If it does, put a zero above

Binary Number	Binary Complement	Octal Complement
101	010	2
0101	1010	12
00101	11010	32
000101	111010	72

Fig. 1.

each of these rightmost zeros. To convert C0B0, you would write:

		0	
C	0	B	0

Continue from the right and write the number 16 above the rightmost nonzero digit of the hex number; write 15 above each of the other digits. Finally, subtract each of the hex digits from the number above it, converting from letters to numbers — or back if needed. C0B0 converts to 3F50 (Example 3).

15	15	16	0
-C (12)	-0	-B (11)	-9
3	15 (F)	5	0

Example 3.

As another example, the two's complement of hex 05 is FB:

15	16
-0	-5
15 (F)	11 (B)

By the way, the two's complement of a two's complement is the original number; the two's complement of FB is 05:

15	16
-F (15)	-B (11)
0	5

Converting Decimal to BCD

Many computers allow calculations to be done in binary coded decimal (BCD) rather than only in binary. (BCD is a combination of binary and decimal.) Converting decimal to BCD is performed in the same way as converting hex to binary: Replace each decimal digit by its *four-bit* binary equivalent from the table. To convert decimal 93, replace 9 by 1001 and 3 by 0011 to get 10010011.

Notice that this result is different from the 01011101 you would get if you converted 93 to binary. In converting to binary, you convert an entire decimal number at once; in converting to BCD, you convert only one digit at a time.

Watch out for one big area of confusion. If you convert decimal 93 to BCD you get 10010011, which looks like binary. Consequently, you might be tempted to convert this "binary" number to hex, by following the standard procedure, to get 93.

This might fool you into thinking that hex 93 is the same as decimal 93, which is not so. The "hex" 93 is not a true hexadecimal number; it is only a form of shorthand that allows you to express the bit pattern 10010011 in a simpler form. If you were employing an assembler that used hex, you might use what looks like hex 93 when you really meant BCD 10010011.

BCD to Decimal

This conversion is the same as that for binary to hex: Arrange the bits in groups of four starting from the right, and convert each group into hex using the table. For instance, BCD 10001001 is grouped into 1000 and 1001, which gives the decimal 89.

In BCD to decimal, you should *never* get the digits A through F. If you do, then the BCD number was wrong. For instance, to convert 00111100, you would get two groups 0011 and 1100. The 0011 converts into a 3, but 1100 converts to C, which is not allowed in decimal. Hence, 00111100 was not a valid BCD number.

So — What's All This Used For?

If all your programming is in BASIC, you will probably never need to know any of this hex magic. But if you do any machine- or assembly-language programming, it will help a lot.

For example, suppose you want to set up a counter at -50 (decimal) and want to convert this to hex. First find +50 in hex: 50 divided by 16 is 3, with a remainder of 2; 3 divided by 16 is 0, with a remainder of 3. So, a decimal +50 is hex 32. Now change this to -50 by finding the two's complement:

15	16
-3	-2
12 (C)	14 (E)

-50 is CE in hex.

Or suppose you want to subtract 2 from some hex number. If your computer does not have a subtract instruction, you can do the same thing by adding a -2. In hex, 2 is 02, and the -2 is found as the two's complement:

15	16
-0	-2
15 (F)	14 (E)

You should add hex FE.

Once you figure it out, hex magic can be fun. ■

Try COMPUTER DIGEST... And Get A Digest of EDP & Scanning News Without Risk

By subscribing to COMPUTER DIGEST, you have the facts you need about computers and optical scanning.

With this information, you can act immediately on new business opportunities. Subscribe and each month you get:

- A 12-page newsletter digesting the important news in the computer and optical scanning field. There is coverage of new equipment, new optical scanning and computer forms, who is expanding, who is getting promoted—a wide variety of facts, all written in concise style, so you can read the reports in minutes.

- Over 50 news reports in each issue covering new equipment, new uses of computers and scanning equipment, new products and software.

- Not only can you find new markets, but the reports will help you solve customer problems, because you will be more knowledgeable about computers and optical scanning.

- It's a timely summary of what is being written about computers and optical scanning in the business press. You save reading time, plus you can be sure of getting hundreds of new items of interest each year.

To subscribe, just complete the form below and mail it back. The newsletter is fully guaranteed. Read 2 issues, see if you can use the information. If you find you don't need COMPUTER DIGEST, just write "cancel" on the invoice we send you, and owe nothing. If you want to continue to read COMPUTER DIGEST, simply pay the $40 invoice we send. We have found this is the best way to introduce businessmen to this newsletter. This way, we take all the risk, so you have nothing to lose by trying this publication.

------------------------------ CLIP AND MAIL ------------------------------

Send to:
COMPUTER DIGEST
North American Building, 401 N. Broad St., Philadelphia, PA 19108

Yes, send me COMPUTER DIGEST, also incorporating OPTICAL SCANNING NEWS. I understand I can look at 2 issues, and if not satisfied, I will owe nothing. I can write "cancel" on the bill, and owe nothing. If I like the newsletter, I will pay the $40 bill I receive.

☐ Bill company for $40
☐ Bill me for $40
☐ Payment enclosed

Name ______________________ Title ______________________

Company ______________________

Address ______________________

City/State/Zip ______________________

Computer Digest is published by North American Publishing Company, leaders in editorial excellence.

KB/5/78

KILOBAUD KLASSROOM NO. 10

Bus Traffic Control

George Young
Sierra High School
Tollhouse CA 93667

In the last session, we covered the majority of the TTL counters and some of the register chips. We performed many experiments with these chips, thus building your background skills in reading circuit diagrams and, I hope, building up your confidence as well. As you can see, the sessions are beginning to get a bit rougher. Hang in there; we will make it yet.

In this session, we will take up decoders, decoding, three-state devices, and how traffic is controlled on the microprocessor data bus.

Introduction

Most of our modern microprocessor chips have 16 address lines providing the capability of selecting 65,535 discrete memory locations. These separate memory locations are referred to as the address space of the microprocessor. Fig. 1 shows the microprocessor and 16 address lines. These are labeled A_0 through A_{15}. 1K of RAM requires ten address lines from the microprocessor to select the 1024 separate memory cells in each RAM chip; so we have drawn the 10 address lines A_0 through A_9 running from the microprocessor to the 1K RAM block.

We are going to draw the 1K RAM block in an unusual fashion. There are actually eight separate RAM chips in the RAM block, and we have drawn them stacked up in order to conserve space. We did not draw eight rectangles in the stack, but the concept of more than one chip is readily conveyed by this diagram.

As shown in Fig. 1, our 1K of RAM will not function; two things are wrong. First, the ten address lines will not drive the address inputs of the 1K RAM block. Microprocessor output pins are capable of driving one TTL load. We are asking each address line to drive eight inputs to the RAM block. Therefore, we must provide buffering on each of the address lines out of the microprocessor. A buffer is a circuit placed between two circuits to provide isolation. We need a buffer on each address line not for isolation, but to increase the drive capability.

The second reason Fig. 1 won't work is that the chip enable ($\overline{CE}$) pins on the RAM chips are *floating*. The $\overline{CE}$ and the small circle on the symbol both indicate that we need an active low enable here to make the RAM function.

In Fig. 2a, we have added noninverting buffers to each address line to provide the drive capability required. Our first idea is to use the A_{10} address line for the $\overline{CE}$ input for the first 1K RAM block. After all, this line will be low for the first 1K of memory space; and when this line goes high, the first 1K RAM block will be de-selected.

We are also introducing another concept in Fig. 2a. The ten address lines, A_0 through A_9, are shown entering a rectangle. Feeding from the rectangle is a widened arrow that goes to each of the RAM blocks. Data lines and address lines are often drawn in this fashion. The broadened line indicates that more than one line is included in the wide line. This saves drawing the individual lines involved and takes less space in the diagram. As long as the idea is understood by everyone, there is no problem, and the diagram is clearer and actually more easily understood.

Furthermore, in Fig. 2a we have added a second 1K RAM block. Our first thought on

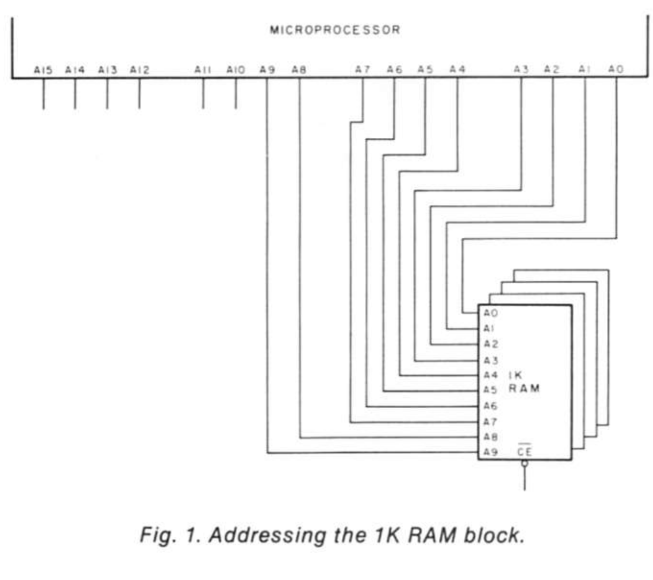
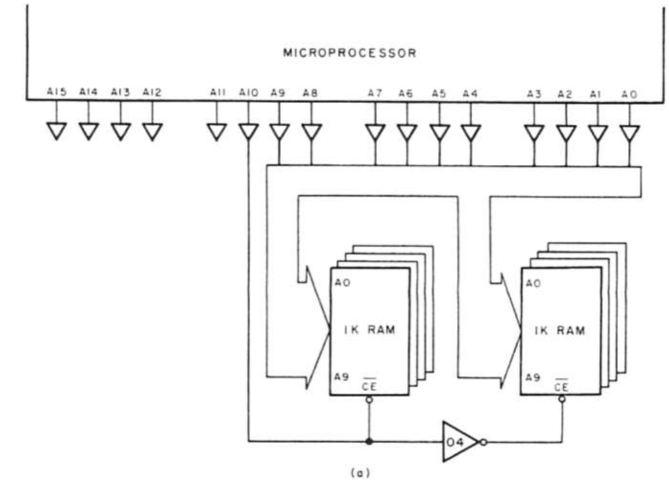


Fig. 1. Addressing the 1K RAM block.



ADDRESSES	A15	A14	A13	A12	A11	A10	
4-1023	0	0	0	0	0	0	1ST 1K
1024-2047	0	0	0	0	0	1	2ND 1K
2048-3071	0	0	0	0	1	0	3RD 1K
3072-4095	0	0	0	0	1	1	4TH 1K
4096-5119	0	0	0	1	0	0	5TH 1K
5120-6143	0	0	0	1	0	1	6TH 1K

(b)

Fig. 2. Adding the second 1K RAM block.

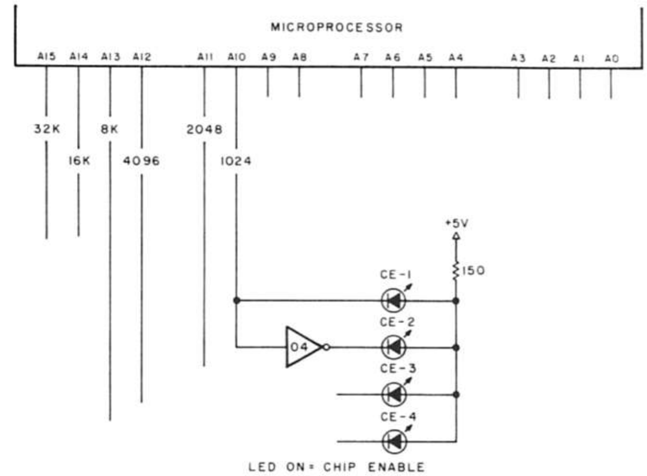


Fig. 3. Experimental setup for decode testing.

handling the second group of $\overline{CE}$ pins on this block is to add an inverter between the $\overline{CE}$ on the first RAM block and the $\overline{CE}$ on the second RAM block. This will work *if* we only have 2K of memory in our system. If we have more RAM or ROM, then an examination of the truth table in Fig. 2b will help us find out why this simple method of enabling the 2K will not work.

The truth table shows that the A_{10} line does indeed start out low for the first 1K of memory space and then is high for the second 1K. But lines 3 and 5 of the table also show the A_{10} line low. Therefore, the first 1K RAM block will be selected every time the A_{10} line goes low. In other words, the single inverter decoder will not do for memory sizes above 2K.

Fig. 3 shows the experimental setup for the design console breadboard and the address lines from the microprocessor. Since we don't have a microprocessor (yet), we'll use this circuit to show how the lines are related; the actual test circuit is shown in Fig. 4a. The chip enable LEDs have been arranged in the circuit to turn on the LED when the CE line goes low.

In Fig. 4 we are attempting to place an equivalent circuit on the console breadboard that will represent what happens with the address lines and the decoding process. Fig. 4a shows the equivalent breadboard circuit for Fig. 2. Note that we are not considering the A_0 through A_9 address lines in the decoding process. These lines are used by each 1K block of memory throughout the address space and are not used in the decoding process for each 1K block.

In order to have a 1K RAM block selected only once in the memory space, we must use some form of decoding. We can use gates and inverters and decode each 1K block in this fashion. Fig. 4b shows this kind of decoder. You can set this circuit up on the console breadboard and use it to decode the four CE lines; but there is an easier way—use a decoder chip. This makes a rather long introduction, but I think that we have the problem fairly well delimited.

Experiment #51
The 7442 Decoder

Problem: How can the address lines of the microprocessor decode the memory chips?

Solution: We will investigate this on the console breadboard. The experiment uses the 7442 decoder, but the 7441, the 7445, the 74145, the 8250 or the 8251 may also be used for this experiment.

Procedure: Refer to Fig. 5. Fig. 5a shows the 7442 pinouts; Fig. 5b shows the 7442 truth table. Notice the row of zeros (lows) traveling diagonally across the truth table . . . this is exactly what we need for chip enable pins. Put the 7442 on the console breadboard (don't forget power and ground). Use four jumper wires to represent the A_{10} through A_{13} address lines. Start with all four inputs to the 7442 grounded. The LED marked CE-1 should be on.

Theory: The 7442 is a one-of-ten (usually written 1:10) decoder. It has four input lines marked A, B, C and D on our diagram. The truth table of Fig. 5b

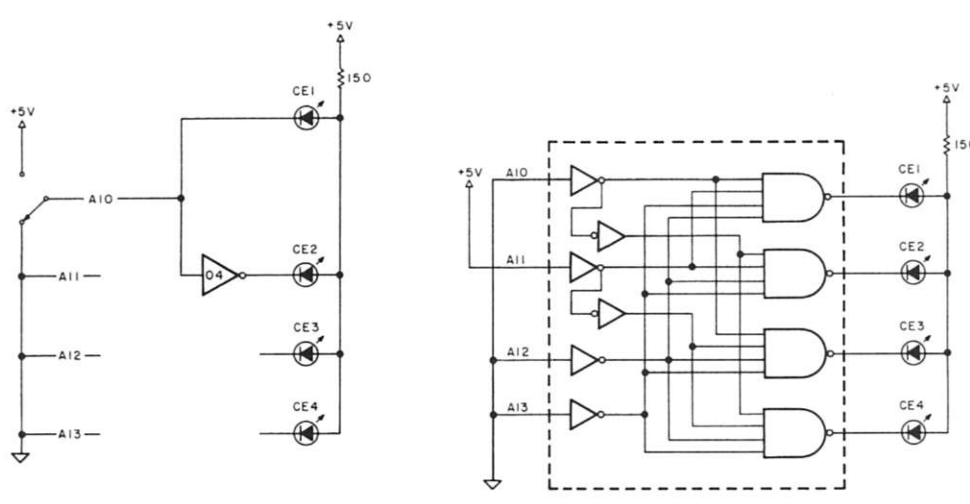


Fig. 4. Delimiting the address decoding problem.

shows that with all inputs low, the 0 output line (pin 1) will be low. This should turn on CE-1. This line would, therefore, go to the first 1K RAM block $\overline{CE}$ pin, and would select that RAM block. (Fig. 5c illustrates decoding 4K of RAM.)

Now take the A_{10} jumper wire high. This should turn on CE-2 and turn off CE-1. This line (from pin 2 on the 7442) would go to the second 1K RAM block and select this RAM block while, at the same time, the first 1K RAM block is de-selected.

If you now encode a binary 2 by taking the A_{11} line high and the A_{10} line low, pin 3 on the 7442 should go low, turning on CE-3 and turning off CE-2. This line from pin 3 on the 7442 would go to the third 1K RAM block and select it while blocks 1 and 2 are de-selected.

Finally, if you encode a binary 3 with both the A_{10} and A_{11} lines high, CE-4 will illuminate and CE-3 will turn off. Pin 4 of the 7442 would go to the fourth 1K RAM block selecting it while the highs on pins 1, 2 and 3 will de-select the first three RAM blocks. Thus, we have a decoder for 4K of memory chips.

But wait, we did not use all the outputs of the 7442. What about the rest of the output pins?

The 7442 may be operated as a 1:4 decoder, 1:8 decoder or 1:10 decoder. To use only the first eight outputs of the 7442, we do not use the D input to the 7442; we leave it grounded. We can then operate the 7442 as a 1:8 decoder and use the eight output pins to decode 8K of RAM. To operate the 7442 as a 1:4 decoder as we just did in the experiment, leave the C and D inputs grounded and operate the 7442 as a 1:4 decoder to decode 4K of address space. We may use all ten out pins of the 7442 and decode 10K of address space with the 7442.

Fig. 6 gives the pin-outs for several more decoder chips.

Experiment #52
The 74154 Decoder Chip

Problem: To decode more than 10K of address space.

Solution: Use a decoder that has more output pins.

Procedure: Refer to Fig. 6e, where the 74154 1:16 decoder is set up in a test circuit. This 24-pin chip was designed for address decoding in computers. It has two enable pins, 18 and 19. Use two jumper wires on these pins to represent the A_{14} and A_{15} address lines. Any binary counter may be used to simulate the A_{10} through A_{13} address lines. Set up the circuit with the 74161 counter chip. Sixteen LEDs are shown monitoring the 74154 output lines.

If you do not have 16 LEDs, then use as many as you can for the test circuit. Remember that the console logic probe may be used for one LED and that you have eight LEDs in the console 7-segment readout. If you have the FND 70 readout, then it will be necessary to drive the segments of the FND 70 through inverter sections since FND 70 requires an active high to turn on each segment. The 74154 will decode 16K of address space.

Experiment #53
The Traffic Cops

Problem: What is all this stuff hung on the data bus lines?

Solution: Let's take a look.

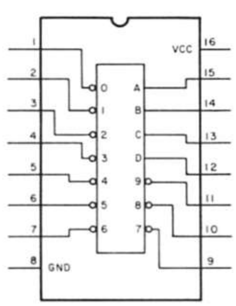
Procedure: Fig. 7 shows the microprocessor chip and its eight data lines. It also shows arrows signifying data traveling both directions on these data lines. During a read cycle, the data is traveling from memory (or input/output devices) into the microprocessor. During a write cycle, data travels from the microprocessor out to external devices. Fig. 7b shows a single data line (D_0) and a pair of open collector NAND gates acting as traffic cops on the data line.

Theory: Assume that the microprocessor is in a memory read cycle. This means that the R/$\overline{W}$ is high. The high on pin 2 of the 7403 will enable this gate, which means the data to be read into the processor will be enabled. This high is also inverted to a low by the inverter section, and the low on pin 5 will disable this gate (taking it out of the circuit for the time being).

Next, the microprocessor is assumed to go into a memory write cycle. The R/$\overline{W}$ goes low, and the low on pin 2 of the 7403 now disables this gate and pin 3 floats on the end of the 2.2k pull-up resistor. The low on the R/$\overline{W}$ line is inverted by the inverter section, and the resulting high output applied to pin 5 will enable this gate. The data to be written into memory (from the processor) will now be enabled onto the data bus. This circuit illustrates how the two-way traffic on the data bus is controlled by the "traffic cops" in the circuit. The R/$\overline{W}$ line and the inverter control the two gates and the direction of the traffic flow.

Fig. 7b is fine for an introduction and example of controlling data on a bus going to and from the processor. However, it isn't practical from a design standpoint (for several reasons). First, the dual-gate configuration would have to be repeated for each data line. This means the R/$\overline{W}$ output from the microprocessor would be driving *eight* gates. You'll recall from an earlier discussion that all of the microprocessor outputs are capable of driving only one TTL gate each. Fig. 7c illustrates a solution to the problem—the addition of an inverter, and a little reconfiguring. Now the R/$\overline{W}$ signal is going into the 7404 (pin 1), which is driving the eight write gates (only one of which is shown).

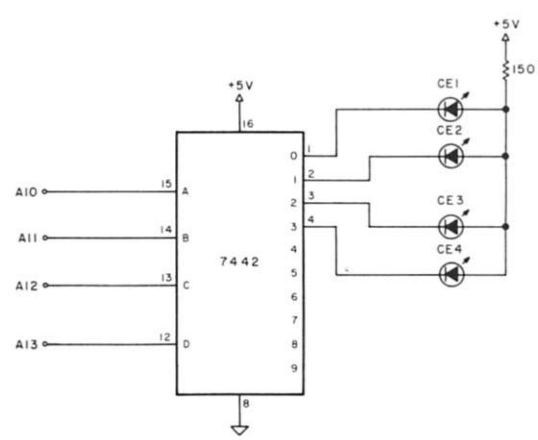
The second, and most important, reason why this circuit is totally unacceptable lies in the use of the 7403 gates for interfacing with the bus. The whole idea behind a bus system is that several devices can be plugged into the bus (i.e., other gates will be tied to the bus further down the line). These additional gates have a "loading effect" on the bus. Without my going into a detailed technical explanation, it will suffice to say that such systems consume a lot of power and are noisy (i.e., have glitches and



(a) 7442 PIN-OUTS

INPUTS				OUTPUTS									
D	C	B	A	0	1	2	3	4	5	6	7	8	9
0	0	0	0	0	1	1	1	1	1	1	1	1	1
0	0	0	1	1	0	1	1	1	1	1	1	1	1
0	0	1	0	1	1	0	1	1	1	1	1	1	1
0	0	1	1	1	1	1	0	1	1	1	1	1	1
0	1	0	0	1	1	1	1	0	1	1	1	1	1
0	1	0	1	1	1	1	1	1	0	1	1	1	1
0	1	1	0	1	1	1	1	1	1	0	1	1	1
0	1	1	1	1	1	1	1	1	1	1	0	1	1
1	0	0	0	1	1	1	1	1	1	1	1	0	1
1	0	0	1	1	1	1	1	1	1	1	1	1	0
1	0	1	0	1	1	1	1	1	1	1	1	1	1
1	0	1	1	1	1	1	1	1	1	1	1	1	1
1	1	0	0	1	1	1	1	1	1	1	1	1	1
1	1	0	1	1	1	1	1	1	1	1	1	1	1
1	1	1	0	1	1	1	1	1	1	1	1	1	1
1	1	1	1	1	1	1	1	1	1	1	1	1	1

(b) 7442 TRUTH TABLE



(c) DECODING 4K OF RAM

Fig. 5. The 7442 decoder.

spikes that can be interpreted as logic ones or zeros). The answer to the problem is to use Tri-state gates for interfacing to a bus.

Tri-state gates, such as the 8T97 shown in Fig. 8a, are either enabled or disabled. When they are enabled by a low on the DISable pins (1 and 15), the outputs will be determined by the logic levels (HI or LO) at the input pins. In other words, the gates are working just like any other gates. When they are disabled (by a high on the DISable line) the gates are effectively disconnected from the bus. The outputs are said to have gone into a high-impedance or open condition and do not present any loading to the bus (i.e., they are disconnected). Fig. 8b is a truth table for the operation of the 8T97 and Fig. 8c illustrates a typical bus interface configuration.

In summary, there are three advantages to using Tri-state gates when you are interfacing to a microcomputer bus (one of which I haven't mentioned before). First, lower power consumption; second, less loading on the bus (thereby maintaining waveform integrity); and finally, higher speed (faster switching from a high to low or vice versa).

Note that the 8T97 is a noninverting buffer and has four sections controlled by one line and two sections controlled by a second line. The two sections may be operated independently of each other. The DM 8097 and the 74367 are also the same type of chip. The 8T97 is more

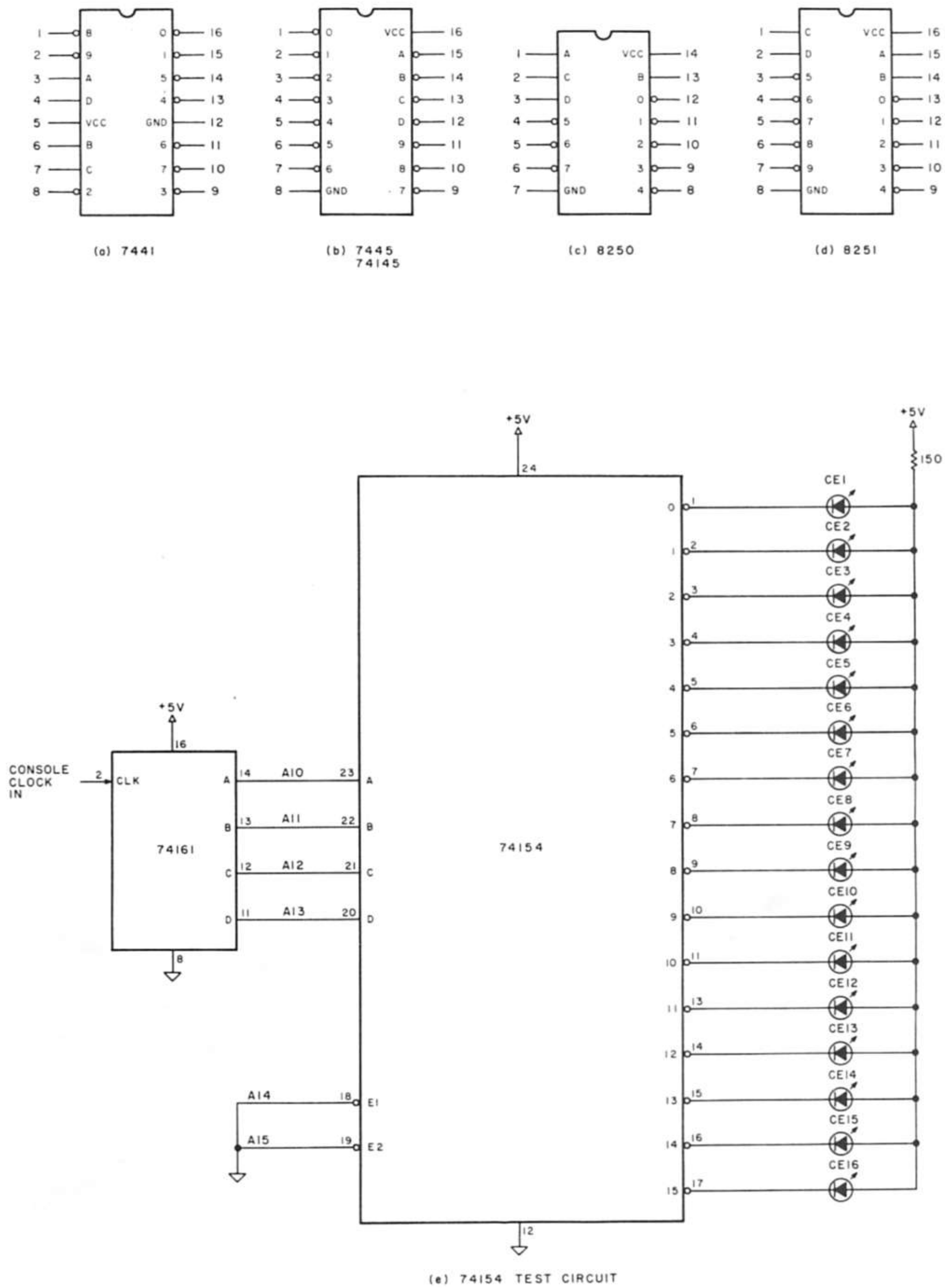


Fig. 6. The 74154 decoder.

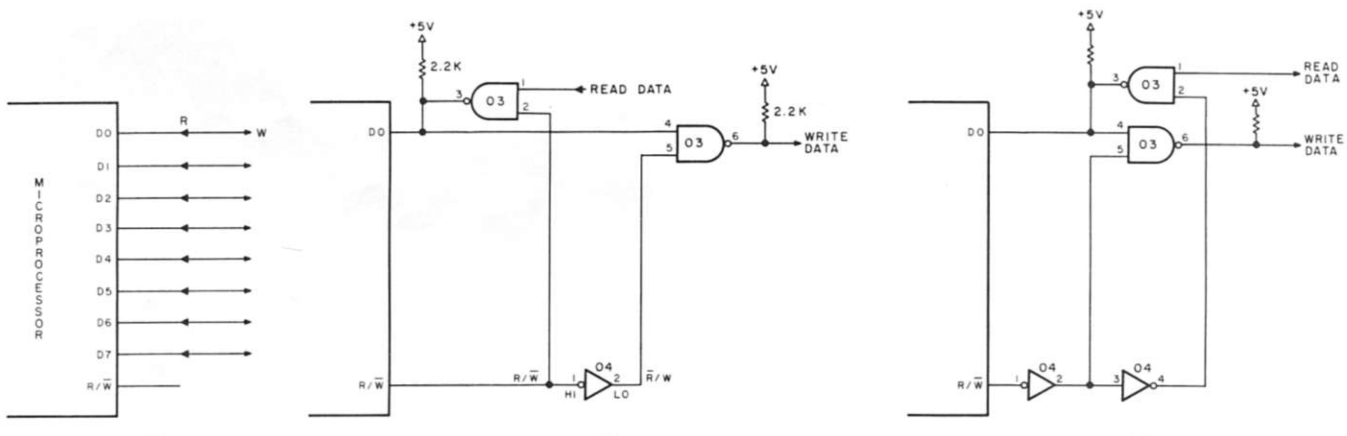


Fig. 7. Traffic control on the data bus.

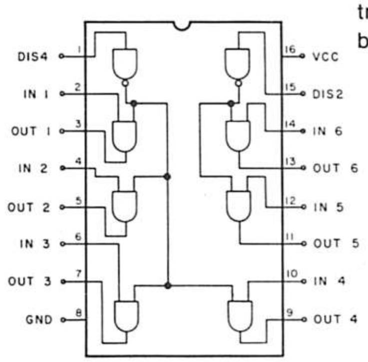
expensive than the others, but my own experience with these chips indicates that the 8T97 has more drive capabilities and proves superior in operation in the circuit . . . justifying its greater cost.

Other chips are becoming available for this buffering job on the data and address buses; I think that soon we may see a new family of microprocessors with the buffers, as well as RAM and ROM, built into the chip. In fact, Intel has a new microprocessor chip, with many of these capabilities built in, which will be second-sourced by Signetics. This points the way that things are heading in the subsequent generation of microprocessor chips.

Preview

We have looked at the microprocessor address bus, how decoding of the address space may be accomplished and how traffic is controlled on the data bus.

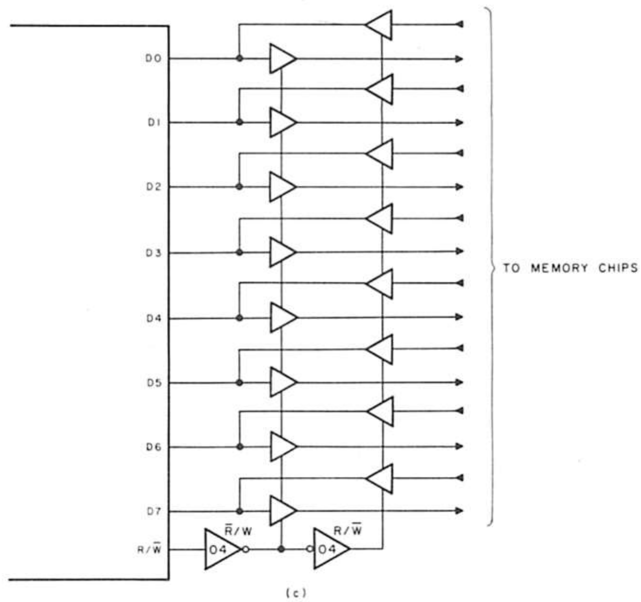
Next time we will turn our attention to the memory chips, both ROM and RAM. Using the 7489 (8225), we will set up 64 bytes of memory on the console breadboard, and also burn a 7488 (8223) PROM on the console. Sierra Electronics, Box 11, Auberry CA 93602, will furnish a package for us of two 8225s and two 8223s for $4 postpaid in the U.S. and Canada. California residents, add 6 percent sales tax.■



(a) 8T97 PIN-OUTS
DM8097
74367

DISABLE DIS2	DISABLE DIS4	INPUT	OUTPUT
0	0	0	0
0	0	1	1
0	1	X	HI Z
1	0	X	HI Z
1	1	X	HI Z

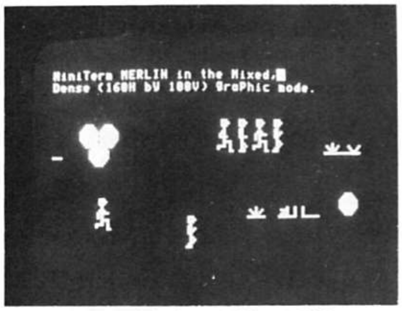
(b) TRUTH TABLE



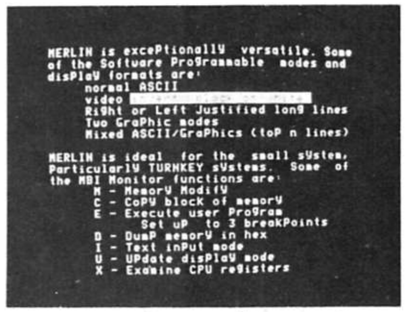
(c)

Fig. 8. 8T97 data bus buffering.

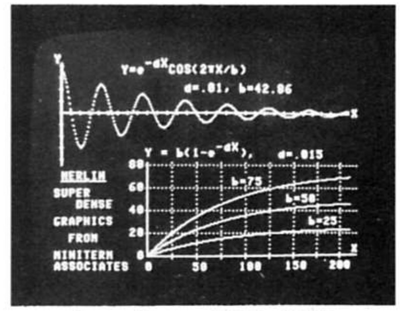
The many faces of MERLIN



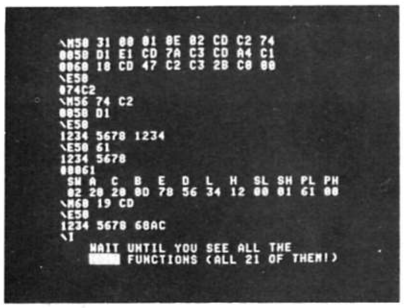
Dense Mode: 160H × 100V
Running Man Patterns



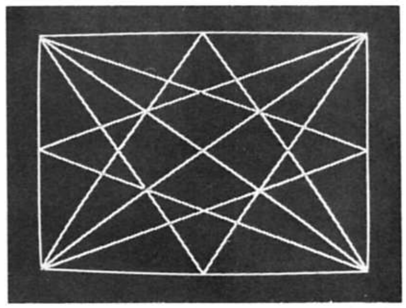
Propaganda

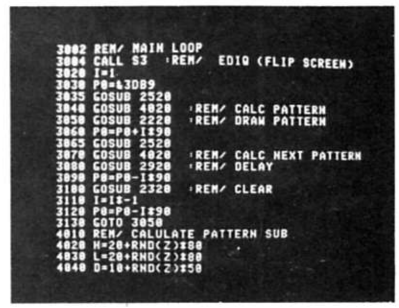


Super Dense: 320H × 200V
Equation Plotting

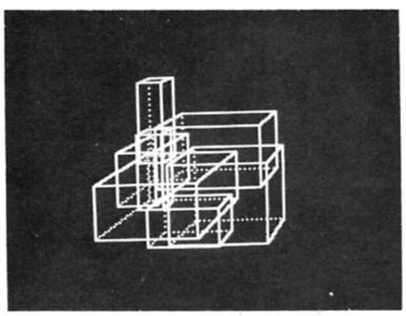


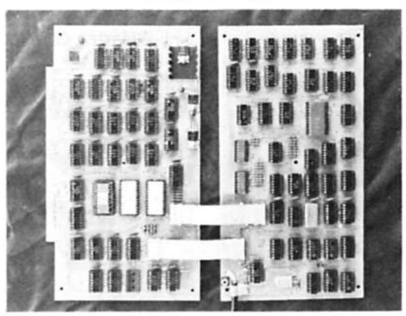
Monitor Debug Usage

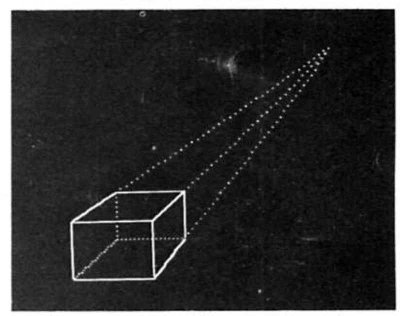

Super Dense: 320H × 200V
Line Drawing



BASIC Program Listing
Output Shown Below


Super Dense: 320H × 200V
3-D Boxes




Super Dense: 320H × 200V
Perspective Drawing

MERLIN (and your S-100 Computer...)
... the graphics development package you've been waiting for.

Do these photos suggest an application in your field? Whether you're into architecture, astrology, music or whatever, you can apply graphics to make your presentations more effective and your work more efficient.

Real Time Plotting
- Heart Rate
- Navigation
- Spectrum Analysis

Complex Equation Plotting
- Stock Market Trends
- Teaching: Mathematics
- Circuit Responses

Fine Line Drawings
- Architecture
- 3-D Projections
- Circuit Layouts

Pattern Movement
- Animation
- Games

MERLIN is also a multipurpose system monitor board with ROM monitor and editor software, parallel keyboard port, and audio cassetts storage, besides being your text and graphics video output device.

Write or call today for your free copy of our new catalog, an index to available application notes, and a list of **MiniTerm** Dealers.

MERLIN is supported by expert technical assistance, a constantly expanding series of application notes, and a newly formed users group.

Assembled and tested **MERLINs** start at $349, $269 for kits. Super Dense add-on, firmware, and cassette interface are extra.

MiniTerm Associates, inc.

Dundee Park, Andover, MA 01810 (617) 470-0525

M40

Expand Your KIM

Part 5: A/D interfacing (for joysticks!)

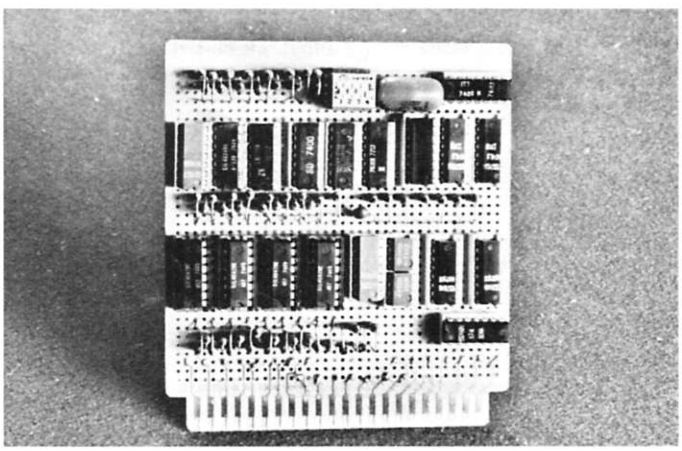


Photo 1. Four channels of A/D, two channels of D/A and an input port for sense switches.

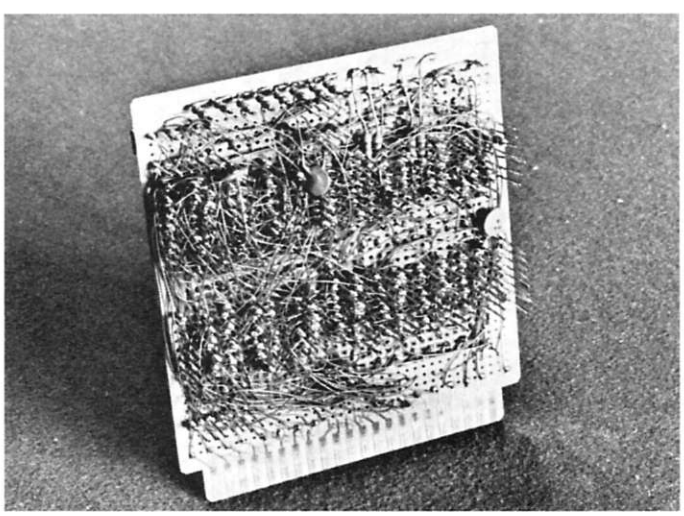


Photo 2. Circuits are wire-wrapped on a 44-pin board.

John Blankenship
datamart, inc.
3001 No. Fulton Dr. N.E.
Atlanta GA 30305

No matter what kind of computer you have, this article can help you add four channels of analog input for a fraction of the cost of other methods I've seen. If you've been building the KIM System, this analog board will complete the project.

I designed the KIM System with several requirements in mind for the analog ports: I required four channels (so that two joysticks could be inter-

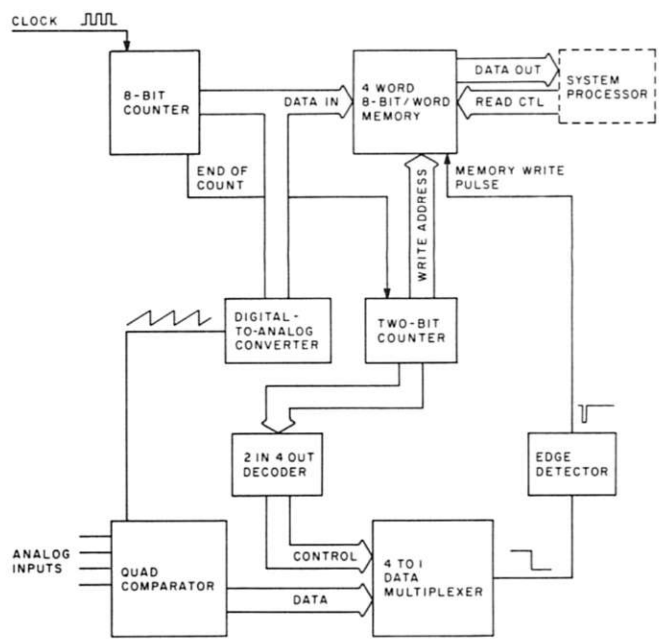


Fig. 1. Block diagram of the A/D converter.

faced), with each sampled often enough to provide reasonable accuracy for use as a video game input device. To make the use of these ports easy, I wanted each to be read as a normal memory. Finally, each of the A/D (analog-to-digital) channels had to be easily switchable to other devices besides the joysticks.

Besides the A/D ports, I also wanted at least two D/A (digital-to-analog) ports to experiment with music, speech synthesis, motor control, etc. I also wanted a port for sense switches to give me a full complement of methods for interfacing with my machine. I combined all these circuits on one board and labeled it *External Interfacing* in my previous articles.

Photos 1 and 2 show the board itself. Although I was able to cram the circuit onto a 4½-inch-square board, I would recommend epoxying a vector board on the top to give more room for the components.

Fig. 1 shows the basic block diagram for the A/D circuits. The four-word memory is one of the major secrets of making this circuitry both inexpensive and easy to use. This memory is made up of two 74LS170 chips composed of four 4-bit words each. I chose these chips because they have separate *read* and *write* controls, thus enabling read and write operations to occur simultaneously.

The A/D circuitry will update each of these memory locations with a number that is proportional to the analog input. The output of the memory chips is connected to the data bus so that they appear as standard memory to the processor.

The eight-bit counter continually generates sequential numbers from 0 to 255. A D/A converter converts these numbers to an analog voltage which, for all practical purposes, is an increasing ramp. This ramp is fed to four comparator circuits that compare the ramp voltage to the analog inputs.

The comparators output a level 1 when the ramp voltage equals the analog input. Since the ramp voltage also equals the number in the eight-bit counter, it is implied that the instant a comparator fires, the eight-bit counter contains the digital equivalent of the analog voltage being applied to that comparator.

The remainder of the circuit has one major function . . . it must decide which comparator fired, and form an address for the four-word memory so the eight-bit counter data can be gated into the appropriate location.

I chose to control the write address with a two-bit counter. Since this counter increments every time the eight-bit counter completes a full cycle, the addresses 0, 1, 2 and 3 are being applied sequentially to the *write* address, and each is held there for the full cycle of the eight-bit counter.

Additionally, this two-bit counter is decoded and used to enable only one of the four comparators (the one corresponding to the write address) at a time. The level change indication from the multiplexer is converted to a narrow pulse and used to activate the write line on the memory chips.

As explained above, the four memory locations are continually, and automatically, refreshed with the digital equivalent of four analog inputs. The processor needs only to read these locations for the latest updates.

Fig. 2 shows the actual schematic of the A/D circuit. The 7493 simply reduces the frequency to a trackable rate. The 1408L8, D/A converter, outputs a current ramp that is converted to a voltage ramp by the 741

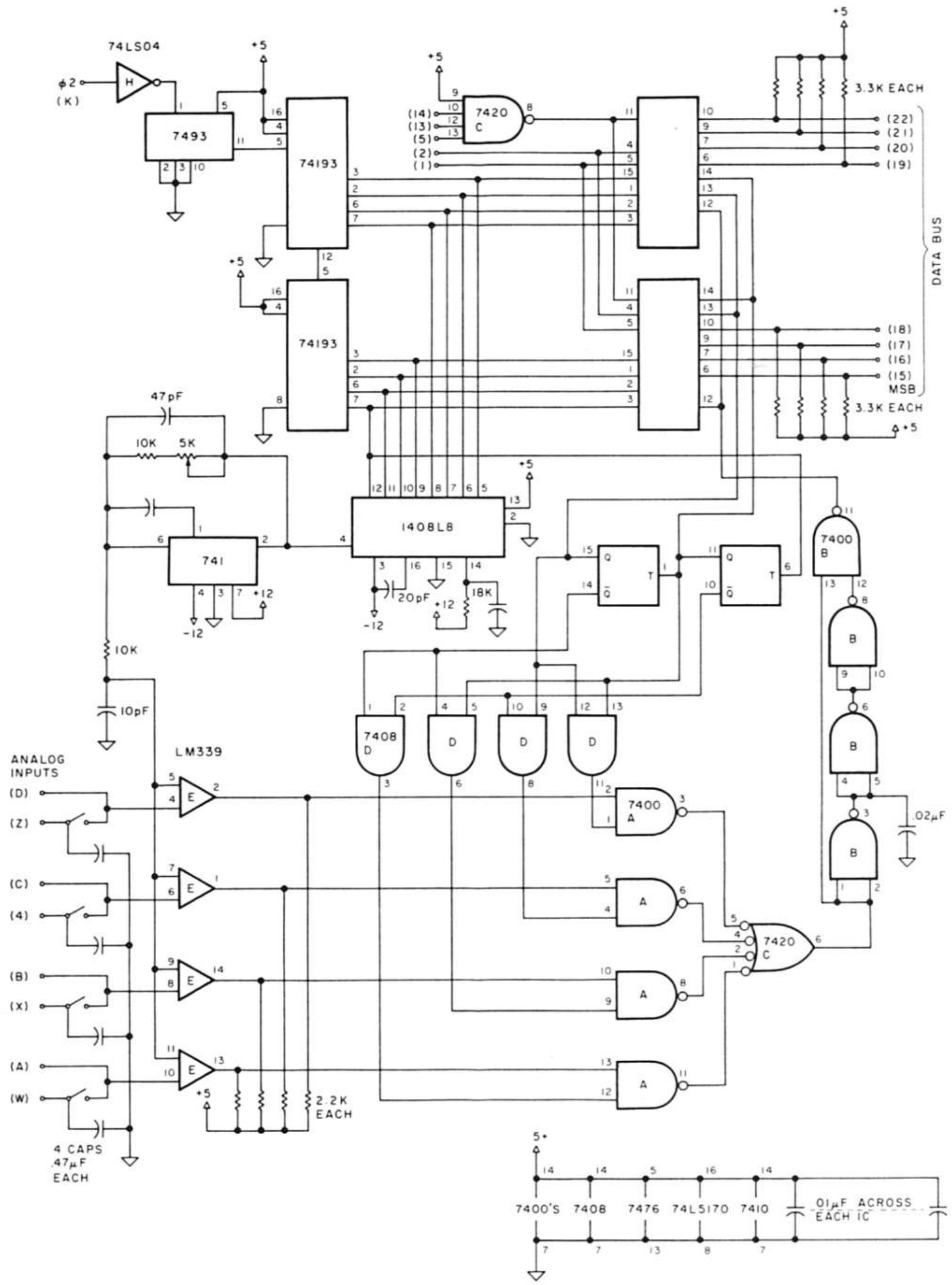


Fig. 2. Schematic of A/D converters.

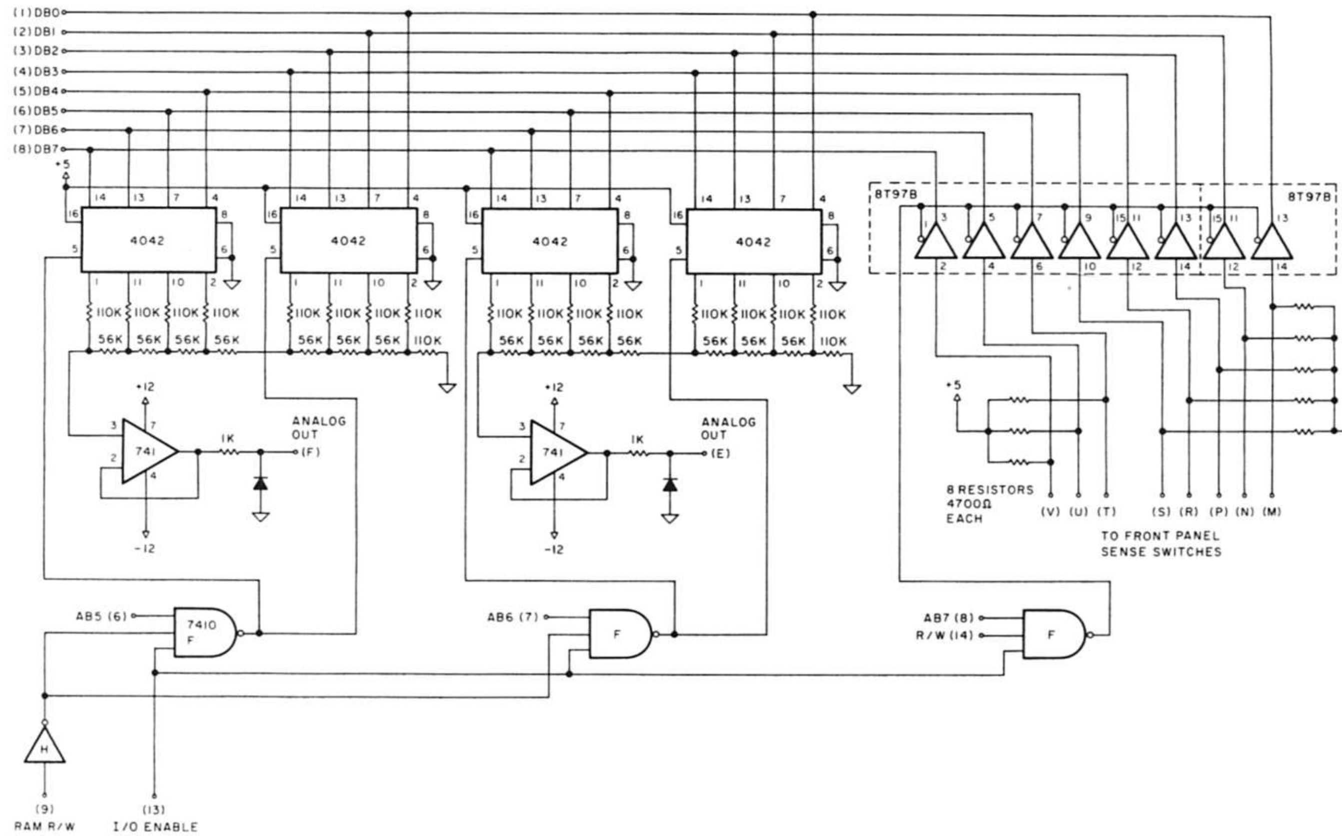


Fig. 3. Schematic of D/A converters and input port for sense switches.

op amp. The 7400 labeled B acts as a one-shot to perform as the edge detector.

Half of the 7420 is used to decode the address bus for processor reads. Address decoding will be discussed in more detail later in this article.

Since the 74LS170s are open collector, rather than Tri-state outputs, pull-up resistors are required for interfacing with the bus. The DIP switch disconnects the joystick inputs. Once they're disconnected, you can input other signals to the converter by way of the backplane jacks (see my earlier articles).

The other two functions, D/A and sense switches, are detailed in Fig. 3. Since I felt that the accuracy of the D/A conversion was not critical, I chose not to use the Motorola D/A converter chip used in the A/D circuit. If I had used the Motorola chip, I would have had to use two eight-bit registers to hold the data, the two D/A chips themselves and a current-to-voltage converter.

I chose to use MOS registers for my output ports. Since MOS gates output *exactly* Vcc and zero volts for their corresponding high and low levels, I used them to drive a resistive ladder directly. Additionally, since MOS chips represent a very small load, they can be hung on the bus without buffering. (Note: MOS chips do represent a relatively large capacitive load, and hanging them directly on the bus is not good practice in expandable systems. In this case, however, I knew exactly what loads I would be dealing with and was able to determine that enough drive capability was present.)

The 741s in Fig. 3 are used as unit gain amplifiers for buffering purposes. The 7410 is used to decode out the address lines to determine which port is being used. The sense switches are connected to the inputs of Tri-state buffers. The outputs of these buffers gate the switch data onto the bus when enabled.

Port Function	Page	Loc
Dazzler Mode control	80	0F
Dazzler ON/OFF, Address	80	0E
Right vertical joystick	80	10
Right horizontal joystick	80	11
Left vertical joystick	80	12
Left horizontal joystick	80	13
Sense switches	80	80
D/A port A	80	20
D/A port B	80	40

Fig. 4. Summary of special addresses used by the KIM-1 System.

A/D D	A	1	AB0
A/D C	B	2	AB1
A/D B	C	3	AB2
A/D A	D	4	AB3
D/A B	E	5	AB4
D/A A	F	6	AB5
−12	H	7	AB6
JS REF. Volt.	J	8	AB7
Ø2	K	9	RAM R/W
Ground	L	10	+12
SS 0	M	11	
SS 1	N	12	+5
SS 2	P	13	I/O ENABLE
SS 3	R	14	W/R
SS 4	S	15	DB7
SS 5	T	16	DB6
SS 6	U	17	DB5
SS 7	V	18	DB4
JS LH	W	19	DB3
JS LV	X	20	DB2
JS RH	Y	21	DB1
JS RV	Z	22	DB0

Fig. 5. Pin-out designations for the external interface board.

In order to better understand the I/O functions, you might reread my article ("Expand Your KIM!" Part 3, *Kilobaud*, February 1978, p. 68) in which I explain how I decoded part of the address lines to indicate an I/O operation, rather than a memory transfer.

All my I/O ports (including the four-word memory used for A/D) are partially enabled by this I/O enable. Since I know how many total ports I designed for, I only partially decoded the low-order address lines. This drastically limited the number of ports available on the KIM System, but the ease of implementation, as well as the reduction in cost, made it well worthwhile.

Fig. 4 summarizes the I/O addresses used uniquely by my system. If you convert these hex addresses to binary, you can see how the appropriate address lines are used to enable each port decoder.

There are only two major differences between input and output decoding. The first is that the R/W or the $\overline{R/W}$ line is used to indicate the direction of the transfer. Second, the *write* pulse for an output port must be coincident with the trailing edge of the Ø2 clock. Again, I refer you to Part 3 of this series for more details.

Fig. 5 shows the pin-out designations for the external interfacing board. These match the mainframe wiring done in Part 2 of this series.

In order to insure that builders of the KIM System fully understand how to utilize the joystick interface, I have included a short program in Fig. 5 that will enable you to draw with the joystick on the TV screen. The sense switches control the colors of the two-color dot that is moved by the joystick.

This program serves a useful function as an educational endeavor, and that's about all. However, I do feel that builders of the KIM System will find it useful as a reference. I have tried to functionally describe each section with comments.

This completes the hardware series on my KIM-1 system, which now contains 17K of RAM and supports both BASIC and FOCAL. I'm also in the process of implementing a new language with an ease of use and a speed of operation somewhere between assembly language and BASIC.

Because my system is to be multilingual, I have chosen to avoid ROM in favor of RAM for all functions except the KIM monitor. I'm also planning several surprises that I hope to share in the future.■

Address	Contents	Label	Mnemonic
00 00	LOC PAGE	DATA	STORE POINTER
	:Set mode and starting address for the dazzler		
02	A9 10	INIT	LDA #$10
04	8D 0F 80		STA MODE
07	A9 90		LDA #$90
09	8D 0E 80		STA BEGADDR
	:Get horizontal joystick position		
0C	AD 11 80	START	LDA JOYHOR
	:Place 4 MSB into 4 LSB and save		
0F	4A		LSR
10	4A		LSR
11	4A		LSR
12	4A		LSR
13	85 00		STA LOC
	:Get vertical joystick position		
15	AD 10 80		LDA JOYVER
	:Check for and set up proper page of screen display		
18	30 07	TOP	BMI BOTTOM
1A	A0 20		LDY #$20
1C	84 01		STY PAGE
1E	4C 25 00		JMP CONT
21	A0 21	BOTTOM	LDY #$21
23	84 01		STY PAGE
	:Remove MSB and keep only the next four		
25	0A	CONT	ASL
26	29 F0		AND #$F0
	:Combine LSB and MSB into one word and save		
28	05 00		ORA LOC
2A	85 00		STA LOC
	:Put color (sense switches) into accumulator		
2C	AD 80 80		LDA SENSE
	:Prepare for an indirect store using 00 and 01 as pointer		
2F	A2 00		LDX #$00
	:Store color		
31	81 00		STA LOC PAGE
	:Begin Again		
33	4C 0C 00		JMP START

Fig. 6. Sample program for drawing on TV using joystick.

Sheila Clarke
CyberGrafix
518 N. Brand
Glendale CA 91203

What's Happening with the IBM Selectric?

Micro Computer Devices has the answer

Art Childs needed a printer for a long time. An IBM Selectric had been Art's and my choice for a couple of years. It was ideal because of its small size and beautiful print quality. However, we both were skeptical about printers available for use with a computer. In most cases, either the typewriter was used and reconditioned or a lot of interface kit assembly was required. Like many computer-users, Art can't afford to risk having an unreliable unit requiring continual maintenance; nor does he have time to assemble a kit.

When we first heard about the SELECTERM, made by Micro Computer Devices (MCD), we were impressed that someone had finally converted a brand new typewriter for microcomputers. Because both IBM and Micro Computer Devices provide warranties for their respective portions of the device, we decided to obtain a SELECTERM.

Art's system consists of an Altair 8800, dual ICOM floppy-disk drive, and an ADM CRT terminal. He uses the 3P + S interface board from Processor Technology. After spending two hours struggling to decipher the board's schematics, which seemed to be written solely for hardware types, Art finally called his engineer friend, Steve Griffis, who came over and had everything running in five minutes. Although the SELECTERM will interface to any microcomputer, what if you can't read the interface board schematics? Micro Computer Devices is providing a solution with specific connecting instructions for each interface available on every computer now being sold.

Art's reaction to the printer was positive from the moment the two large cartons were delivered. One carton held the Selectric and the other contained the electronics package. He was impressed with the packing, which held the units solidly with formed foam to prevent damage caused in shipping. Opening the flap of the carton, Art uncovered a sheet that said STOP, with complete unpacking and typewriter assembly instructions. Art, in too big a hurry, merely made a mental note that instructions were there and, consequently, ran into a little trouble securing the cover latches of the typewriter. (Sometimes I wonder if anyone reads anything before making panic calls to the manufacturer.)

He was also impressed with the documentation and the SELECTERM's acceptance of ASCII. With no conversion necessary, Art began writing a driver. It took him five minutes, using assembly language for FDOS-III. He said the only difference between this printer and another line printer driver or hard-copy output driver is that you might have to put out some nulls after tabs and line feed. But it was simple for him to write the nulls into the driver. Fig. 1 shows the driver for the 8080 and 3P + S.

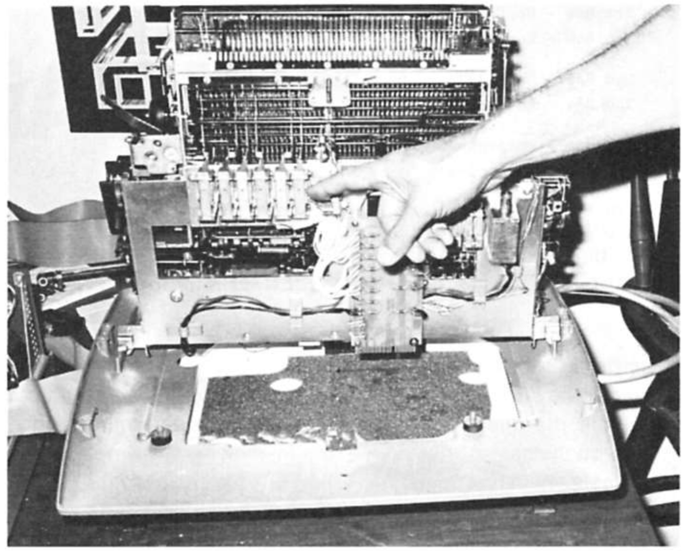
Art commented: "The IBM print quality is nice. And I like the fact that I can change type fonts. Putting the whole thing together—removing it from the cartons to putting the cover on the typewriter and hooking up the cables—was a half-hour task. The fact that it requires one parallel port makes it easy. If you have only one serial port, which is often the case, you'll usually lose it to your print device. Writing the driver and integrating it into the software completed the process. All in all, it was very easy; everything's been done for you. The unit runs very cool, the electronics box is barely warm to the touch after running consistently for about three hours, and it runs cooler than the typewriter itself."

I'm using the SELECTERM to prepare this article for *Kilobaud*; I am inputting the text in the computer, from first draft to the final, edited version. It's a pleasure to know I don't have to retype this thing two or three times before I get it right. The advantages of the SELECTERM are only evident when I begin to use it. For example, the sales literature doesn't tell me how to input uppercase and lowercase letters with a terminal that has only uppercase. So MCD owner Shelly Howard pointed out the ADM has switches beneath the nameplate. Setting the LC EN switch enables me to input uppercase and lowercase for printer output. I did discover, however, that the switch must always be returned to the UC position after using the Insert mode of the text editor. After that little switch is flipped, the CRT may only see uppercase characters, but when I hit shift for uppercase characters, the printer outputs caps where they should be—just like using a typewriter.

After using the SELECTERM for a couple of weeks, Art and I ran into difficulty getting clear print—then it jammed. The problem was a loose motor mount. Because the typewriter portion was under warranty, IBM service came out and fixed it at no charge.

How It All Began

To find out how his product came about I spent some time talking with Shelly Howard. Like many other small-scale manufacturers, Shelly knew relatively little about microcomputers two and a half years ago. In fact, he was preparing his thesis for his PhD on an IBM Selectric. After gathering sufficient research data, he wanted it compiled through a computer and output on a Selectric that matched the type of his own typewriter. He was told by two computer outfits that IBM had discontinued making its I/O device. He was forced to either scrap his original plans or buy his own computer. Assuming the cost of ownership would be prohibitive, he searched and discovered the world of microcomputers. He also discovered Don Lancaster's *TV Typewriter Cookbook*.



If we lift the typewriter up off the baseplate, we see the electronics added to convert the typewriter to a printer.

Now They Tell Me!

Although he followed the book's instructions to the letter, Shelly failed to get a unit up and running. He later discovered the book had been based on theory only; no one in Lancaster's organization had actually put the theory to practice. By now Shelly was too committed to back out, so he decided to start over with the help of two design engineers, Steve Garner and Jimmy Carter (no, another one).

Months of design development, field testing and improvements resulted in production of a printer with all parts—the baseplate, actuators, coils, transformer and linkages—manufactured by MCD. Finally, the design was approved by IBM. That's why IBM service will come and fix your printer if anything goes wrong; you can also buy yearly service agreements from IBM after the warranty expires. For this reason, MCD will not sell the SELECTERM in kit form. IBM has only approved the factory assembled and tested model.

```
0000                  ;         ROUTINE TO DRIVE SELECTERM WITH 8080 AND 3P+S
0000                  ;
0000   DB04     LO:     IN      4         ;GET STATUS
0002   E601             ANI     1         ;MASK
0004   CA0000           JZ      LO        ;NOT READY
0007   79               MOV     A,C       ;GET CHAR
0008   D306             OUT     6         ;OUTPUT
000A   FE09             CPI     9         ;WAS IT A TAB?
000C   CA1C00           JZ      LOTAB     ;YES
000F   FE08             CPI     8         ;NO - BACKSPACE?
0011   CA1700           JZ      LOLF      ;YES
0014   FE0A             CPI     0AH       :NO - LINE FEED?
0016   C0               RNZ               ;NO - RETURN
0017                  ;
0017   0E00     LOLF:   MVI     C,0       ;OUTPUT A NULL
0019   C30000           JMP     LO        ;AND RETURN
001C                  ;
001C   C5       LOTAB:  PUSH    B
001D   010004           LXI     B,400H    ;4 NULLS
0020   CD0000   LOTB1:  CALL    LO        ;OUTPUT
0023   05               DCR     B         ;LAST ONE?
0024   C22000           JNZ     LOTB1     ;NO
0027   C1               POP     B         ;YES - RESTORE B
0028   C9               RET               ;AND RETURN
0029                  ;
0029   0000             END
TOTAL ERRORS=00
```

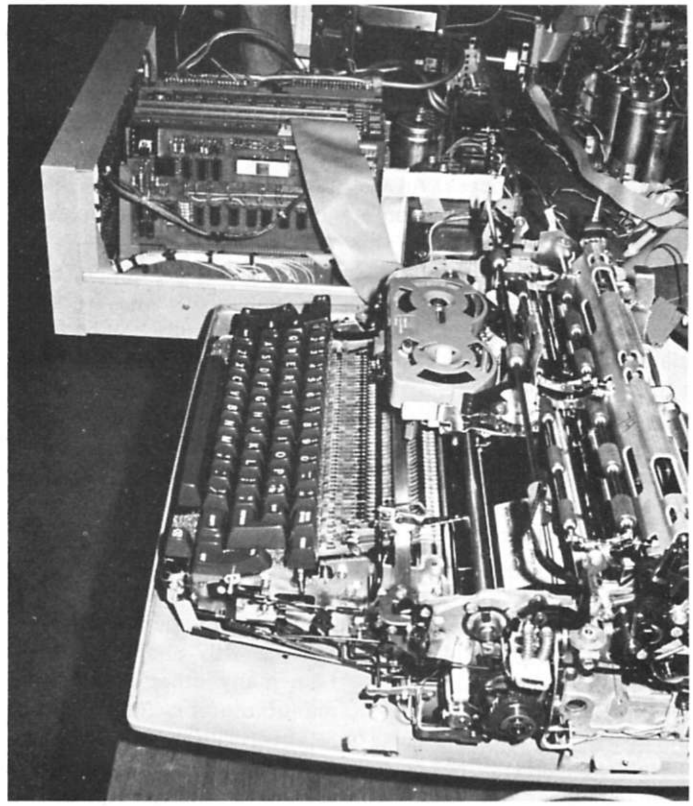
Fig. 1. ICOM 8080/Z-80 Reloc-Macro Assembler Ver. 1.0.

In Full Swing

First shipments of the SELECTERM were made in August 1977; currently about three per day are delivered to dealers. The target is five per day, but the cash-flow situation is tough with MCD in a continual fiscal squeeze. Though IBM sanctioned the design, MCD is treated like any other individual consumer, as far as open credit goes. When you buy in quantity, with no quantity discount, at the same price I paid for my

1st Row - Uppercase:	!	@	#	$	%	¢	&	*	(	)	_	+
1st Row - Lowercase:	1	2	3	4	5	6	7	8	9	0	-	=
2nd Row - Uppercase:	Q	W	E	R	T	Y	U	I	O	P	¼	
2nd Row - Lowercase:	q	w	e	r	t	y	u	i	o	p	½	
3rd Row - Uppercase:	A	S	D	F	G	H	J	K	L	:	"	
3rd Row - Lowercase:	a	s	d	f	g	h	j	k	l	;	'	
4th Row - Uppercase:	Z	X	C	V	B	N	M	,	.	?		
4th Row - Lowercase:	z	x	c	v	b	n	m	,	.	/		

Fig. 2. ASCII character set for SELECTERM output device.



When the typewriter cover is off, the SELECTERM looks about like another Selectric II. Here it sits alongside Art's Altair 8800 with cabling interface to the 3P + S.

Selectric II, a lot of bucks are going out the door at one time. To handle the dilemma, MCD sells through dealers only, on a COD basis. Because requests have been made by some manufacturers, the firm wants to produce OEM versions to specification. Shelly will probably find investors, or perhaps release MCD for acquisition by another company. But he loves what he's doing: selling and delivering SELECTERMs to dealers across the country.

Competition

Presently, only one other company in the country sells an IBM Selectric printer with ASCII encoding. Other companies offer used Selectrics complete with interfacing. Even reconditioned units will not qualify for the IBM Service Agreement.

If you're looking for a good printer, this could be it. But take heed that 15 characters per second may not be fast enough. Long listings could take hours. For most home computerists, however, speed may not be a determining factor in making a printer selection. And the benefits are numerous: All the basic features of the printer include the special typing element, tab command, back space, vertical tab, bell, serial and parallel interfacing, cable sets and software in PROM within the electronics. Also included is a special ASCII typing element that IBM has produced to MCD specifications. Fig. 2 shows an output of the character set.

The price of $1750 appears prohibitive, until you consider that you'll be using an extremely well-designed unit that will last for years—type fonts are changed at will, no special paper is needed, IBM ribbon is easy to order, and service is virtually hassle-free.

Options

The same extras as those offered by IBM, including dual pitch and correcting feature, can be ordered for your SELECTERM. MCD has developed a noise-reduction feature (recommended if you live in a residential neignborhood).

Tractor-feed platen and RS-232 interface are also being offered as options.

After using the SELECTERM a great deal for two months now, Art and I are definitely convinced that we did a good thing for his computer. And a nice plus is that we now have a second typewriter—that is, when it's not being used with the computer. ■

The Top-Down Approach

with some practical examples

Dr. Lance A. Leventhal
PO Box 1258
Rancho Santa Fe CA 92067

In *Kilobaud* No. 14 ("Why Structured Programming?" p. 84), I discussed structured programming, a method for making the logic of large programs simple and repetitive, thereby making them easier to debug and test. But a further problem in writing large programs is how to put sections of the programs together. This article describes a widely used method called top-down design, by which the programmer starts with an overall outline of the program and proceeds to steadily describe each section in greater detail, debugging and testing along the way in an integrated manner.

Modular Programming

Obviously, a large program can only be written by dividing it into sections. No one (I hope) would simply write the entire program and then see if it worked. Clearly, a better idea is to write a small section, see if that works, correct it, write another small section, and so on. This procedure is known as *modular programming* and the sections of the program are called *modules.*

Some typical modules in an overall accounting, game, word-processing or instructional program might be: I/O routines, file-handling routines, mathematical calculations, string-handling routines, table searches, sorting routines, table lookup and list processing.

The advantages of modular programming are clear.

1. You can check the modules individually and be sure they work properly. Thus, you can assume that any errors in the overall program are in the connections or the supervisor program.

2. You can build a library of modules that will be useful in other programs. Many of the previously mentioned modules will be needed frequently.

3. You can use modules that you have previously developed, found in books or magazine articles, or borrowed from friends. You can also use modules such as file handlers, code converters and I/O handlers that comprise part of your monitor or operating system.

4. You can plan program development and have a reasonable idea of how much progress you have made and what the major stumbling blocks are.

5. You can eliminate many simple errors at an early stage.

Modular programming has serious disadvantages, though. Somehow, the modules never quite seem to fit together at the end. Different modules may use different registers, memory locations or subroutines. Some may wipe out results that others need or not use data that others provide. Module integration often turns out to be a big task you must struggle with after everything seems to be done.

The problem of integrating modules is independent of the problem of testing and debugging them. The modules may all work separately, but still not work together. The catch is that the original debugging and testing checks the workings of the module out of context (i.e., all by itself rather than as part of a complete program).

In fact, debugging and testing a module in isolation can be quite difficult. A game program, for example, may consist of the following modules: (1) determine initial conditions, (2) read and check proposed move (see if it is valid), (3) determine new conditions, (4) print status.

But how can you write the routine that reads and checks the proposed move unless you know the previous state of the game and can see the new state? How will you be able to tell if the MOVE module is working properly? Typically, you will have to either manually enter the required data and examine the results or write special programs to perform those tasks. These special programs (sometimes called *driver programs*) can save a lot of manual effort; however, they introduce extra work and may act quite differently from the real routines for which they substitute. (Note that you don't save the driver programs; you throw them away when the job is done.)

Clearly, the problem of combining modules is even more serious in large commercial programming projects. Not only can the number of modules in a project be very large, but also many programmers may be involved in writing them. Now the problem is to integrate modules written by people with different styles, different levels of expertise, different documentation methods and different interpretations of tasks.

Top-down Design

Most commercial programming shops now use some version of top-down design. This method differs from the more traditional bottom-up design (see Fig. 1) in which the specific modules are written before they are integrated into more complex programs. Top-down design (see Fig. 2) proceeds as follows:

1. The overall supervisor program is written, debugged and tested. Major subprograms are replaced by *program stubs* that may produce the answer to a selected problem, record the entry or do nothing at all.

2. Each stub is then similarly

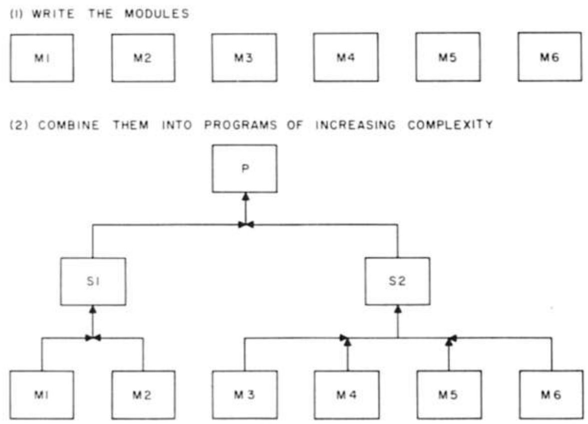


Fig. 1. The procedure for bottom-up design.

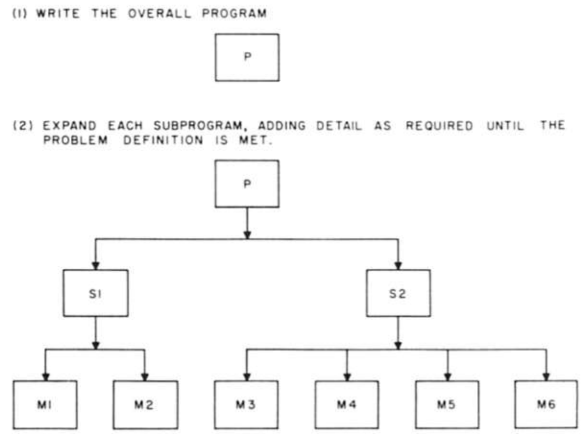


Fig. 2. The procedure for top-down design.

expanded, with debugging and testing occurring at each step.

Advantages of Top-down Design

The advantages of top-down design are:

- It modularizes debugging, testing and integration, as well as coding (the writing of instructions).
- It allows subprograms to be debugged and tested in the actual environment of the entire program. No special debugging and testing programs (or drivers) are needed to provide data or to interpret results.
- It results in overall program logic being checked first. This often means that the programmer can immediately discover and eliminate inconsistencies and misconceptions that otherwise may be very difficult to find and correct (after all the modules have been written).
- It provides a systematic framework for program development and testing. It gives the programmer a firm idea of how much of the task has been accomplished.

Disadvantages of Top-down Design

Of course, like all methods, top-down design has disadvantages. Among these are:

- A suitable program stub may be difficult to write, particularly if it must appear in many different places and produce many different incidental effects.
- The top-down expansion may not mesh well with hardware or already existing software.
- Errors in the overall program can have catastrophic effects on the entire project. Often critical design decisions must be made early before you know what problems exist (or will be created) at the lower levels.

Furthermore, top-down design assumes a simple program structure with independent subsections (i.e., a tree structure, as shown in Figs. 1 and 2). Some programs (perhaps even most) can logically be constructed in that manner. But there is no proof that all, or even most, programs can be. Often programs have interconnections at all levels that defy simple analysis.

Of course, top-down design is no panacea; it provides neither rules nor guidance for: (1) dividing programs into modules that can be written independently of other modules; (2) writing the modules (here, structured programming comes into play); (3) defining or using data structures . . . in many situations, the structure of the data may be more important and more difficult to determine than the structure of the program.

But top-down design does provide a systematic framework, rather than a haphazard approach. This framework has been shown to significantly increase programmer productivity in the commercial world. Furthermore, it seems to result in programs that have clearer logic and are easier to test, debug, extend and use. Of course, programmers should never disdain a little bottom-up design where that method permits better utilization of hardware, existing software or other resources. The aim of programming is to produce programs that work, not to follow the tenets of one methodology or another.

Much of what we have said so far about top-down design is vague. Now let us see how it works in a real example.

The Vote Analysis Program

The purpose of this program is to count ballots and print the totals in decreasing order—starting with the candidates who received the most votes. C is the number of candidates, and the ballots are coded as follows:

0—a blank ballot (no vote for any candidate).
1 to C—vote for the indicated candidate.
C + 1—vote for a write-in candidate.
C + 2—illegal vote (two or more candidates marked).
C + 3—special marking for last (dummy) ballot.

Fig. 3 shows the initial program flowchart. The important variables are: N (I)—number of votes for candidate I, V—total number of votes, M (I)—candidate numbers for rank-ordering.

We have not tried here to make the programs particularly efficient or to make the I/O realistic. Rather, we have tried to show how program development proceeds, starting with an overall skeleton program and continuing through ever-increasing levels of detail. The language is a simple version of BASIC that should run on most computers.

Initial Program

Fig. 4 contains the initial program listing. The three major sections of the program—counting, ordering and output—have been replaced by program stubs that simply mark those sections that have been entered. We can test the overall program logic by entering a value for the number of candidates, C, and running the

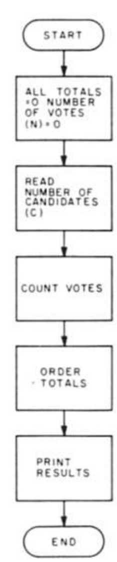


Fig. 3. Initial flowchart for the vote-analysis program.

```
LIST
10   DIM N(20),M(20)
15   REM NUMBER ØF  VØTES (V) = 0
20   LET V= 0
25   REM GET NUMBER ØF CANDIDATES (C)
30   PRINT "NUMBER ØF CANDIDATES = ";
35   INPUT C
40   REM CLEAR ALL VØTE CØUNTERS
45   FØR I= 1 TØ C+ 2
50   LET N(I)= 0
55   NEXT I
60   REM CØUNT VØTES
65   GØSUB 1000
70   REM ØRDER VØTE TØTALS
75   GØSUB 2000
80   REM ØUTPUT TØTALS
85   GØSUB 3000
999   END
1000   REM VØTE CØUNTING PRØGRAM
1010   PRINT "ATTEMPTED VØTE CØUNTING"
1020   RETURN
2000   REM TØTAL ØRDERING PRØGRAM
2010   PRINT "ATTEMPTED ØRDERING"
2020   RETURN
3000   REM ØUTPUT RØUTINE
3010   PRINT "REACHED ØUTPUT RØUTINE"
3020   RETURN
9999   END
RUN
NUMBER ØF CANDIDATES = ? 0
ATTEMPTED VØTE CØUNTING
ATTEMPTED ØRDERING
REACHED ØUTPUT RØUTINE
READY
```

Fig. 4. Initial listing for the vote-analysis program. All the subprograms are left as unexpanded stubs.

program (note the RUN results at the bottom). In fact, there was a slight error initially caused by the omission of the final END statement. This error was quickly corrected before any stubs were expanded.

The First Level of Expansion

Fig. 5 is the flowchart of the expanded vote-counting pro-

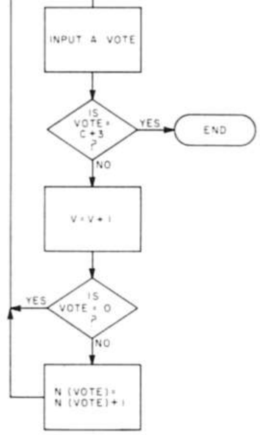


Fig. 5. Flowchart for the vote-counting subprogram.

gram. Here there are three cases to consider:

1. The last ballot (marked with the number C + 3) is not counted in the totals.
2. Blank ballots (marked by zero) are included in the total number of votes but are not credited to any category.
3. Other ballots must be credited to the appropriate category (i.e., to a candidate, write-in category or improperly marked category).

Fig. 6 contains the BASIC program with the vote-counting stub expanded. We checked this program with the data in Example 1 (see the results at the bottom of Fig. 6).

Fig. 7 contains the BASIC program with the output stub expanded. This program was also checked with cases 1 and 2. Note the added statement

3020 IF C = 0 THEN 3045

This correction means that if there are no candidates, the program does not print headings, a list of candidates or vote totals. Note that the case

```
LIST
10   DIM N(20),M(20)
15   REM NUMBER ØF  VØTES (V) = 0
20   LET V= 0
25   REM GET NUMBER ØF CANDIDATES (C)
30   PRINT "NUMBER ØF CANDIDATES = ";
35   INPUT C
40   REM CLEAR ALL VØTE CØUNTERS
45   FØR I= 1 TØ C+ 2
50   LET N(I)= 0
55   NEXT I
60   REM CØUNT VØTES
65   GØSUB 1000
70   REM ØRDER VØTE TØTALS
75   GØSUB 2000
80   REM ØUTPUT TØTALS
85   GØSUB 3000
999   END
1000   REM VØTE CØUNTING PRØGRAM
1005   REM FETCH NEXT VØTE (J)
1010   PRINT "NEXT VØTE IS";
1015   INPUT J
1020   REM DØNE IF VØTE IS ENDING MARK (C+3)
1025   IF J=C+ 3 THEN 1065
1030   REM ADD VØTE TØ TØTAL (V)
1035   LET V=V+ 1
1040   REM IGNØRE VØTE IF BALLØT UNMARKD (J=0)
1045   IF J= 0 THEN 1010
1050   REM ADD VØTE TØ APPRØPRIATE TØTAL
1055   LET N(J)=N(J)+ 1
1060   GØTØ 1010
1065   RETURN
2000   REM TØTAL ØRDERING PRØGRAM
2010   PRINT "ATTEMPTED ØRDERING"
2020   RETURN
3000   REM ØUTPUT RØUTINE
3010   PRINT "REACHED ØUTPUT RØUTINE"
3020   RETURN
9999   END

RUN
NUMBER ØF CANDIDATES = ? 0
NEXT VØTE IS? 3
ATTEMPTED ØRDERING
REACHED ØUTPUT RØUTINE
READY

RUN
NUMBER ØF CANDIDATES = ? 1
NEXT VØTE IS? 1
NEXT VØTE IS? 4
ATTEMPTED ØRDERING
REACHED ØUTPUT RØUTINE
READY
```

Fig. 6. Listing for the vote-analysis program with the vote-counting subprogram expanded.

without a candidate, although it seems useless, is by no means an uncommon situation in real elections, particularly at the local level. The results from this expanded program are in Fig. 8.

Fig. 9 is a flowchart for the first expansion of the rank-ordering routine. The idea is to keep interchanging pairs of elements until all pairs are in the correct order (i.e., largest number first). Flag F is cleared initially and set to 1 if an interchange is performed. So, if F = 1 at the end of a pass through the list, another pass is necessary. If F = 0 at the end, the list must be in order. Although this may appear an unsophisticated sorting method, it is perfectly acceptable for short lists like the ones handled by this program. The number of candidates in an election rarely exceeds ten. Note that no sorting is necessary if there is only one candidate or are none.

Fig. 10 is the BASIC program with the ordering routine ex-

```
LIST
10   DIM N(20),M(20)
15   REM NUMBER ØF VØTES (V) = 0
20   LET V= 0
25   REM GET NUMBER ØF CANDIDATES (C)
30   PRINT "NUMBER ØF CANDIDATES = ";
35   INPUT C
40   REM CLEAR ALL VØTE CØUNTERS
45   FØR I= 1 TØ C+ 2
50   LET N(I)= 0
55   NEXT I
60   REM CØUNT VØTES
65   GØSUB 1000
70   REM ØRDER VØTE TØTALS
75   GØSUB 2000
80   REM ØUTPUT TØTALS
85   GØSUB 3000
999   END
1000   REM VØTE CØUNTING PRØGRAM
1005   REM FETCH NEXT VØTE (J)
1010   PRINT "NEXT VØTE IS";
1015   INPUT J
1020   REM DØNE IF VØTE IS ENDING MARK (C+3)
1025   IF J=C+ 3 THEN 1065
1030   REM ADD VØTE TØ TØTAL (V)
1035   LET V=V+ 1
1040   REM IGNØRE VØTE IF BALLØT UNMARKED (J=0)
1045   IF J= 0 THEN 1010
1050   REM ADD VØTE TØ APPRØPRIATE TØTAL
1055   LET N(J)=N(J)+ 1
1060   GØTØ 1010
1065   RETURN
2000   REM TØTAL ØRDERING PRØGRAM
2010   PRINT "ATTEMPTED ØRDERING"
2020   RETURN
3000   REM ØUTPUT RØUTINE
3005   PRINT "NUMBER ØF CANDIDATES = ";C
3010   PRINT "NUMBER ØF VØTES = ";V
3015   REM SKIP CANDIDATE TØTALS IF NØ CANDIDATES
3020   IF C= 0 THEN 3045
3025   PRINT "CANDIDATE NUMBER     VØTE TØTAL"
3030   FØR I= 1 TØ C
3035   PRINT TAB( 5),I,TAB( 25),N(I)
3040   NEXT I
3045   PRINT "NUMBER ØF WRITE-INS = ";N(C+ 1)
3050   PRINT "NUMBER ØF IMPRØPER BALLØTS = ";N(C+ 2)
3055   RETURN
9999   END
```

Fig. 7. Listing for the vote-analysis program with the vote-counting and output subprograms expanded.

```
RUN
NUMBER ØF CANDIDATES = ? 0
NEXT VØTE IS? 3
ATTEMPTED ØRDERING
NUMBER ØF CANDIDATES =  0
NUMBER ØF VØTES =  0
NUMBER ØF WRITE-INS =  0
NUMBER ØF IMPRØPER BALLØTS =  0
READY

RUN
NUMBER ØF CANDIDATES = ? 1
NEXT VØTE IS? 1
NEXT VØTE IS? 4
ATTEMPTED ØRDERING
NUMBER ØF CANDIDATES =  1
NUMBER ØF VØTES =  1
CANDIDATE NUMBER     VØTE TØTAL
     1                   1
NUMBER ØF WRITE-INS =  0
NUMBER ØF IMPRØPER BALLØTS =  0
READY

RUN
NUMBER ØF CANDIDATES = ? 2
NEXT VØTE IS? 1
NEXT VØTE IS? 1
NEXT VØTE IS? 2
NEXT VØTE IS? 5
ATTEMPTED ØRDERING
NUMBER ØF CANDIDATES =  2
NUMBER ØF VØTES =  3
CANDIDATE NUMBER     VØTE TØTAL
     1                   2
     2                   1
NUMBER ØF WRITE-INS =  0
NUMBER ØF IMPRØPER BALLØTS =  0
READY
```

Fig. 8. Results from the program of Fig. 7.

CASE 1. NO CANDIDATES, NO VOTES
C = 0
V = 3 (ENDING MARKER)
CASE 2. ONE CANDIDATE, ONE VOTE
C = 0
V = 1
V = 4 (ENDING MARKER)

Example 1.

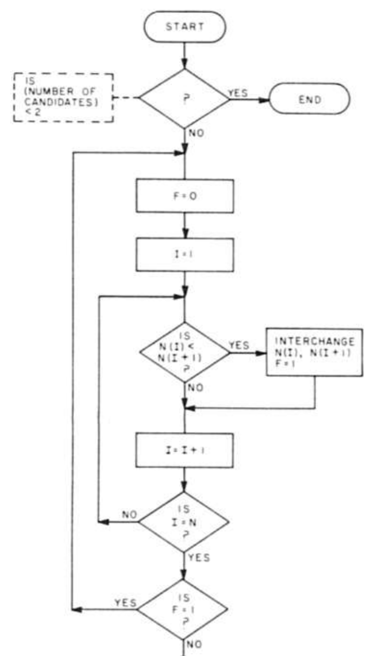


Fig. 9. Flowchart for the rank-ordering subprogram.

panded. Note that the interchange subroutine is left as a program stub. It will be expanded later. For some simple cases for checking this program, see Example 2. Fig. 11 shows the results from this program. Note that an interchange was attempted in Case 4, but not in Case 3.

The Second Level of Expansion

Fig. 12 shows the program with the interchange stub expanded. Statement 3035 now prints the identification number M(I), which is interchanged, but statements 2010 and 2033 had to be changed to give a value to M(I) when there is only one candidate.

Fig. 12 also contains a further expansion of the ordering routine (see flowchart in Fig. 13) to handle more efficiently the simple, but common, case where there are only two candidates. Further expansions could check for erroneous values of number of candidates

```
LIST
10    DIM N(20),M(20)
15    REM NUMBER ØF VØTES (V) = 0
20    LET V= 0
25    REM GET NUMBER ØF CANDIDATES (C)
30    PRINT "NUMBER ØF CANDIDATES = ";
35    INPUT C
40    REM CLEAR ALL VØTE CØUNTERS
45    FØR I= 1 TØ C+ 2
50    LET N(I)= 0
55    NEXT I
60    REM CØUNT VØTES
65    GØSUB 1000
70    REM ØRDER VØTE TØTALS
75    GØSUB 2000
80    REM ØUTPUT TØTALS
85    GØSUB 3000
999   END
1000  REM VØTE CØUNTING PRØGRAM
1005  REM FETCH NEXT VØTE (J)
1010  PRINT "NEXT VØTE IS";
1015  INPUT J
1020  REM DØNE IF VØTE IS ENDING MARK (C+3)
1025  IF J=C+ 3 THEN 1065
1030  REM ADD VØTE TØ TØTAL (V)
1035  LET V=V+ 1
1040  REM IGNØRE VØTE IF BALLØT UNMARKED (J=0)
1045  IF J= 0 THEN 1010
1050  REM ADD VØTE TØ APPRØPRIATE TØTAL
1055  LET N(J)=N(J)+ 1
1060  GØTØ 1010
1065  RETURN
2000  REM TØTAL ØRDERING PRØGRAM
2005  REM NØ ØRDERING NECESSARY IF ZERØ ØR ØNE CANDIDATES
2010  IF C< 2 THEN 2085
2015  REM ASSIGN MARKERS TØ CANDIDATES FØR SØRTING
2020  FØR I= 1 TØ C
2025  LET M(I)=I
2030  NEXT I
2035  REM SØRT VØTE TØTALS
2040  LET F= 0
2045  FØR I= 1 TØ C- 1
2050  REM CHECK IF TØTALS ARE IN ØRDER
2055  IF N(I)>=N(I+ 1) THEN 2070
2060  REM IF ØUT ØF ØRDER, INTERCHANGE PAIR
2065  GØSUB 2500
2070  NEXT I
2075  REM DØ ANØTHER PASS IF ANY INTERCHANGES ØCCURRED
2080  IF F= 1 THEN 2040
2085  RETURN
2500  REM INTERCHANGE TØTALS, MARKERS FØR ØRDERING
2510  PRINT "ATTEMPTED INTERCHANGE"
2520  RETURN
3000  REM ØUTPUT RØUTINE
3005  PRINT "NUMBER ØF CANDIDATES = ";C
3010  PRINT "NUMBER ØF VØTES = ";V
3015  REM SKIP CANDIDATE TØTALS IF NØ CANDIDATES
3020  IF C= 0 THEN 3045
3025  PRINT "CANDIDATE NUMBER     VØTE TØTAL"
3030  FØR I= 1 TØ C
3035  PRINT TAB( 5),I,TAB( 25),N(I)
3040  NEXT I
3045  PRINT "NUMBER ØF WRITE-INS = ";N(C+ 1)
3050  PRINT "NUMBER ØF IMPRØPER BALLØTS = ";N(C+ 2)
3055  RETURN
9999  END
```

Fig. 10. Listing of vote-analysis program with all subprograms expanded by one level.

```
RUN
NUMBER ØF CANDIDATES = ? 0
NEXT VØTE IS? 3
NUMBER ØF CANDIDATES =  0
NUMBER ØF VØTES =  0
NUMBER ØF WRITE-INS =  0
NUMBER ØF IMPRØPER BALLØTS =  0
READY


RUN
NUMBER ØF CANDIDATES = ? 1
NEXT VØTE IS? 1
NEXT VØTE IS? 4
NUMBER ØF CANDIDATES =  1
NUMBER ØF VØTES =  1
CANDIDATE NUMBER     VØTE TØTAL
     1                    1
NUMBER ØF WRITE-INS =  0
NUMBER ØF IMPRØPER BALLØTS =  0
READY


RUN
NUMBER ØF CANDIDATES = ? 2
NEXT VØTE IS? 1
NEXT VØTE IS? 2
NEXT VØTE IS? 2
NEXT VØTE IS? 5
ATTEMPTED INTERCHANGE
NUMBER ØF CANDIDATES =  2
NUMBER ØF VØTES =  3
CANDIDATE NUMBER     VØTE TØTAL
     1                    1
     2                    2
NUMBER ØF WRITE-INS =  0
NUMBER ØF IMPRØPER BALLØTS =  0
READY
```

Fig. 11. Results from the program of Fig. 10.

CASE 3. TWO CANDIDATES. THREE VOTES (2 FOR NUMBER 1, 1 FOR NUMBER 2)

C = 2
V = 1
V = 1
V = 2
V = 5 (ENDING MARKER)

CASE 4. TWO CANDIDATES, THREE VOTES (1 FOR NUMBER 1, 2 FOR NUMBER 2)

C = 2
V = 1
V = 2
V = 2
V = 5 (ENDING MARKER)

Example 2.

(less than zero or more than the program can handle) and erroneous data (values that are undefined). Other expansions could check for ties, handle cases where more than one vote is allowed (e.g., vote for four of the above) and identify the ballots on which write-ins were marked.

Conclusion

Top-down design is a method for designing, debugging and testing large programs. It requires the programmer to start with the overall program logic and to continue expanding subprograms until the task is fully defined. Each level is checked in its actual working environment before the next level is attempted. Thus, integration of modules and system-level debugging and testing are performed throughout program development rather than all at the end. Program stubs replace unexpanded programs or modules at each level. Top-down design is a systematic approach to writing large programs. Personal computer users should carefully consider its use when attempting complex projects.■

References

1. J. K. Hughes and J. J. Michtom, *A Structured Ap-*

```
LIST
10   DIM N(20),M(20)
15   REM NUMBER ØF VØTES (V) = 0
20   LET V= 0
25   REM GET NUMBER ØF CANDIDATES (C)
30   PRINT "NUMBER ØF CANDIDATES = ";
35   INPUT C
40   REM CLEAR ALL VØTE CØUNTERS
45   FØR I= 1 TØ C+ 2
50   LET N(I)= 0
55   NEXT I
60   REM CØUNT VØTES
65   GØSUB 1000
70   REM ØRDER VØTE TØTALS
75   GØSUB 2000
80   REM ØUTPUT TØTALS
85   GØSUB 3000
999  END
1000  REM VØTE CØUNTING PRØGRAM
1005  REM FETCH NEXT VØTE (J)
1010  PRINT "NEXT VØTE IS";
1015  INPUT J
1020  REM DØNE IF VØTE IS ENDING MARK (C+3)
1025  IF J=C+ 3 THEN 1065
1030  REM ADD VØTE TØ TØTAL (V)
1035  LET V=V+ 1
1040  REM IGNØRE VØTE IF BALLØT UNMARKED (J=0)
1045  IF J= 0 THEN 1010
1050  REM ADD VØTE TØ APPRØPRIATE TØTAL
1055  LET N(J)=N(J)+ 1
1060  GØTØ 1010
1065  RETURN
2000  REM TØTAL ØRDERING PRØGRAM
2005  REM DØNE IF NØ CANDIDATES
2010  IF C= 0 THEN 2085
2015  REM ASSIGN MARKERS TØ CANDIDATES FØR SØRTING
2020  FØR I= 1 TØ C
2025  LET M(I)=I
2030  NEXT I
2031  REM NØ ØRDERING NECESSARY IF ØNLY ØNE CANDIDATE
2033  IF C= 1 THEN 2085
2035  REM SØRT VØTE TØTALS
2036  REM HANDLE CASE ØF ØNLY TWØ CANDIDATES SEPARATELY
2038  IF C= 2 THEN 2090
2040  LET F= 0
2045  FØR I= 1 TØ C- 1
2050  REM CHECK IF TØTALS ARE IN ØRDER
2055  IF N(I)>=N(I+ 1) THEN 2070
2060  REM IF ØUT ØF ØRDER, INTERCHANGE PAIR
2065  GØSUB 2500
2070  NEXT I
2075  REM DØ ANØTHER PASS IF ANY INTERCHANGES ØCCURRED
2080  IF F= 1 THEN 2040
2085  RETURN
2090  REM ØRDER TØTALS FØR TWØ CANDIDATES ØNLY
2095  REM NØ PRØBLEM IF ALREADY IN ØRDER
2100  IF N( 1)>=N( 2) THEN 2120
2105  REM IF ØUT ØF ØRDER, INTERCHANGE
2110  LET I= 1
2115  GØSUB 2500
2120  RETURN
2500  REM INTERCHANGE TØTALS, MARKERS FØR ØRDERING
2505  REM MARK THAT INTERCHANGE ØCCURRED (F=1)
2510  LET F= 1
2515  REM INTERCHANGE TØTALS
2520  LET T=N(I)
2525  LET N(I)=N(I+ 1)
2530  LET N(I+ 1)=T
2535  REM INTERCHANGE MARKERS
2540  LET T=M(I)
2545  LET M(I)=M(I+ 1)
2550  LET M(I+ 1)=T
2555  RETURN
3000  REM ØUTPUT RØUTINE
3005  PRINT "NUMBER ØF CANDIDATES = ";C
3010  PRINT "NUMBER ØF VØTES = ";V
3015  REM SKIP CANDIDATE TØTALS IF NØ CANDIDATES
3020  IF C= 0 THEN 3045
3025  PRINT "CANDIDATE NUMBER      VØTE TØTAL"
3030  FØR I= 1 TØ C
3035  PRINT TAB( 5),M(I),TAB( 25),N(I)
3040  NEXT I
3045  PRINT "NUMBER ØF WRITE-INS = ";N(C+ 1)
3050  PRINT "NUMBER ØF IMPRØPER BALLØTS = ";N(C+ 2)
3055  RETURN
9999  END
```

Fig. 12. Listing of vote-analysis program with improved rank-ordering subprogram. The subprogram now handles the case of two candidates more efficiently.

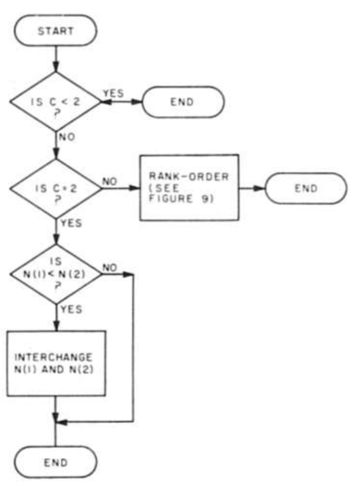


Fig. 13. Flowchart of the improved rank-ordering subprogram.

```
RUN
NUMBER ØF CANDIDATES = ? 2
NEXT VØTE IS? 1
NEXT VØTE IS? 1
NEXT VØTE IS? 2
NEXT VØTE IS? 5
NUMBER ØF CANDIDATES =  2
NUMBER ØF VØTES =  3
CANDIDATE NUMBER    VØTE TØTAL
    1                    2
    2                    1
NUMBER ØF WRITE-INS =  0
NUMBER ØF IMPRØPER BALLØTS =  0
READY


RUN
NUMBER ØF CANDIDATES = ? 2
NEXT VØTE IS? 1
NEXT VØTE IS? 2
NEXT VØTE IS? 2
NEXT VØTE IS? 5
NUMBER ØF CANDIDATES =  2
NUMBER ØF VØTES =  3
CANDIDATE NUMBER    VØTE TØTAL
    2                    2
    1                    1
NUMBER ØF WRITE-INS =  0
NUMBER ØF IMPRØPER BALLØTS =  0
READY
```

Fig. 14. Results from the program of Fig. 12.

proach to Programming, Prentice-Hall, Englewood Cliffs NJ, 1977.

2. E. Yourdon, *Techniques of Program Structure and Design*, Prentice-Hall, Englewood Cliffs NJ, 1975.

3. B. W. Kernighan and P. J. Plauger, *The Elements of Programming Style*, McGraw-Hill, NY, 1974.

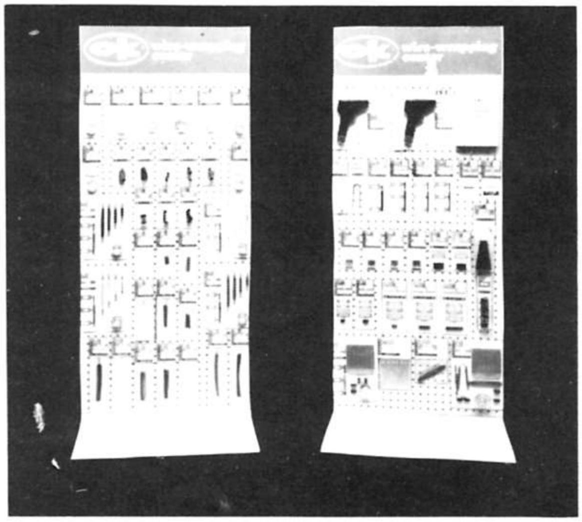
4. E. J. Miller, Jr., and G. E. Lindamood, "Structured Programming: Top-down Approach," *Datamation*, December 1973, pp. 55-57.

5. R. W. Ulrickson, "Solve Software Problems Step by Step," *Electronic Design*, January 18, 1977, pp. 54-58.

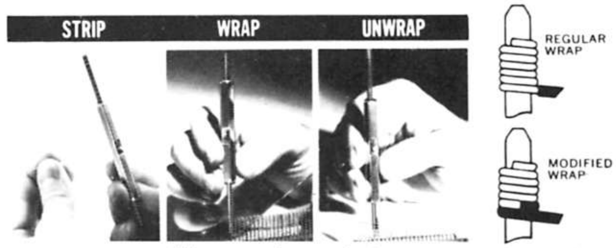
6. L. A. Leventhal, *8080A/8085 Assembly Language Programming*, Osborne and Associates, Berkeley CA, 1978.

ok®

wire wrapping center



for quality electronic parts and tools.



HOBBY WRAP TOOL

Wire-wrapping, stripping, unwrapping tool for AWG 30 on .025 (0,63mm) Square Post.

Regular Wrap	WSU-30	$6.95
Modified Wrap	WSU-30M	$7.95

WIRE-WRAPPING TOOL

For .025" (0,63mm) sq. post "MODIFIED" wrap, positive indexing, anti-overwrapping device.

For AWG 30	BW-630	$34.95*
For AWG 26-28	BW-2628	$39.95*

Bit for AWG 30	BT-30	$3.95
Bit for AWG 26-28	BT-2628	$7.95

*USE "C" SIZE NI-CAD BATTERIES (NOT INCLUDED)

WIRE-WRAPPING KITS

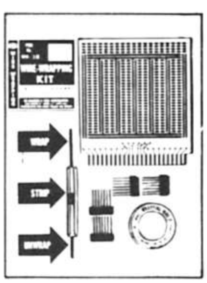
Contains: Hobby Wrap Tool WSU-30, (50 ft.) Roll of wire Prestripped wire 1" to 4" lengths (50 wires per package) stripped 1" both ends.

Wire Wrapping Kit (Blue)	WK 2 B	$12.95
Wire Wrapping Kit (Yellow)	WK 2 Y	$12.95
Wire Wrapping Kit (White)	WK 2 W	$12.95
Wire Wrapping Kit (Red)	WK 2 R	$12.95

ROLLS OF WIRE

Wire for wire-wrapping AWG-30 (0.25mm) KYNAR* wire, 50 ft. roll, silver plated, solid conductor, easy stripping.

30 AWG Blue Wire 50ft Roll	R 30B 0050	$1 98
30 AWG Yellow Wire 50ft Roll	R 30Y 0050	$1 98
30 AWG White Wire 50ft Roll	R 30W 0050	$1 98
30 AWG Red Wire 50ft Roll	R 30R 0050	$1 98



WIRE-WRAPPING KIT

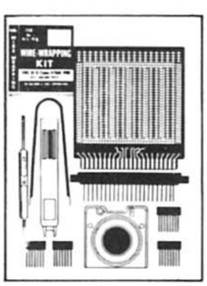
Contains: Hobby Wrap Tool WSU-30, Roll of wire R-30B-0050, (2) 14 DIP's, (2) 16 DIP's and Hobby Board H-PCB-1.

Wire-Wrapping Kit	WK-3B (Blue)	$16.95

WIRE DISPENSER

- With 50 ft. Roll of AWG 30 KYNAR* wire-wrapping wire.
- Cuts the wire to length.
- Strips 1" of insulation.
- Refillable (For refills, see above)

Blue Wire	WD-30-B	$3.95
Yellow Wire	WD-30-Y	$3.95
White Wire	WD-30-W	$3.95
Red Wire	WD-30 R	$3.95



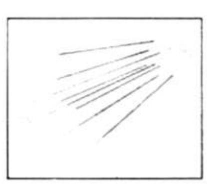
WIRE-WRAPPING KIT

Contains: Hobby Wrap Tool WSU-30 M, Wire Dispenser WD-30-B, (2) 14 DIP's, (2) 16 DIP's, Hobby Board H-PCB-1, DIP/IC Insertion Tool INS-1416 and DIP/IC Extractor Tool EX-1

Wire-Wrapping Kit	WK-4B (Blue)	$25.99

PRE CUT PRE STRIPPED WIRE

Wire for wire-wrapping, AWG-30 (0.25mm) KYNAR* wire, 50 wires per package stripped 1" both ends.



30 AWG blue Wire 1" Long	30 B 50 010	$ 99
30 AWG Yellow Wire 1" Long	30 Y 50 010	$ 99
30 AWG White Wire 1" Long	30 W 50 010	$ 99
30 AWG Red Wire 1" Long	30 R 50 010	$ 99
30 AWG Blue Wire 2" Long	30 B 50 020	$1 07
30 AWG Yellow Wire 2" Long	30 Y 50 020	$1 07
30 AWG White Wire 2" Long	30 W 50 020	$1 07
30 AWG Red Wire 2" Long	30 R 50 020	$1 07
30 AWG Blue Wire 3" Long	30 B 50 030	$1 16
30 AWG Yellow Wire 3" Long	30 Y 50 030	$1 16
30 AWG White Wire 3" Long	30 W 50 030	$1 16
30 AWG Red Wire 3" Long	30 R 50 030	$1 16
30 AWG Blue Wire 4" Long	30 B 50 040	$1 23
30 AWG Yellow Wire 4" Long	30 Y 50 040	$1 23
30 AWG White Wire 4" Long	30 W 50 040	$1 23
30 AWG Red Wire 4" Long	30 R 50 040	$1 23
30 AWG Blue Wire 5" Long	30 B 50 050	$1 30
30 AWG Yellow Wire 5" Long	30 Y 50 050	$1 30
30 AWG White Wire 5" Long	30 W 50 050	$1 30
30 AWG Red Wire 5" Long	30 R 50 050	$1 30
30 AWG Blue Wire 6" Long	30 B 50 060	$1 38
30 AWG Yellow Wire 6" Long	30 Y 50 060	$1 38
30 AWG White Wire 6" Long	30 W 50 060	$1 38
30 AWG Red Wire 6" Long	30 R 50 060	$1 38

* KYNAR PENNWALT

MINIMUM ORDER $25.00, SHIPPING CHARGE $1.00, N.Y. CITY AND STATE RESIDENTS ADD TAX

OK MACHINE & TOOL CORPORATION

3455 Conner St., Bronx, N.Y. 10475 ■ (212) 994-6600 ■ Telex 125091

The North Star Floppy System

an 11-year-old can build it!

Howie DiBlasi
Director, Vocational Education
Lake Havasu High School
Lake Havasu AZ 86403

Hi. My name is Mark; I am 11 years old. I just finished a North Star Floppy Disk Kit. It was easy; I really made it. And guess what? It worked the first time I hooked it up!"

Mark looked at me and smiled. He was really proud of himself, and I was too. If an 11-year-old can put the North Star Kit together, so can you.

"Hey, Dad, am I going to be rich and famous because I put the North Star together and you are writing about me in *Kilobaud*?"

I laughed. Rich? No. Famous? No. Proud and satisfied? Yes.

Here We Go

I ordered the North Star Kit and received it in a week from the Byte Shop in Phoenix. When I opened the box and examined the contents, I was impressed with the quality of the circuit boards and parts. All the parts were there, and complete instructions were included.

After looking over the instruction manual, I had my son read it to see if he understood what to do. He said, "No sweat," and at that point I decided to let him go ahead and build the kit.

Printed Matter

Four instruction manuals came with the kit: (1) Minifloppy Diskette Storage Drive OEM Manual; (2) North Star Disk Operating System Manual; (3) North Star BASIC Manual; (4) North Star MICRO-DISK SYSTEM MDS-A Instruction Manual.

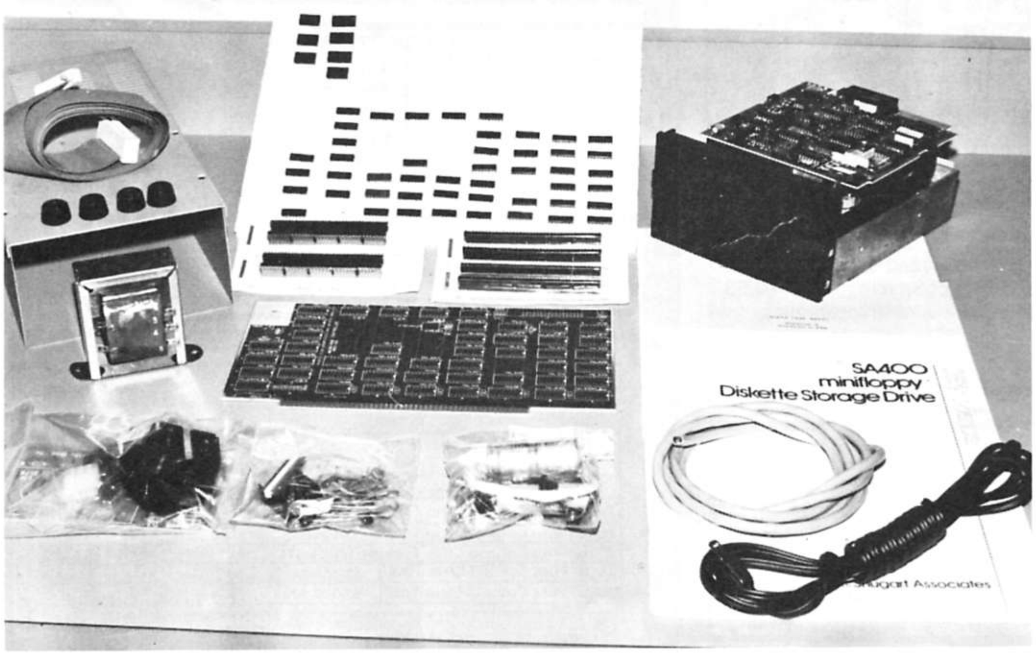
The instruction manual is divided into three sections: theory of operation, assembly instructions and system integration and schematics. The manuals are all well written and detail numerous situations and how to set things up. It was a pleasure to read through and understand the material. Right on, North Star!

Assembly

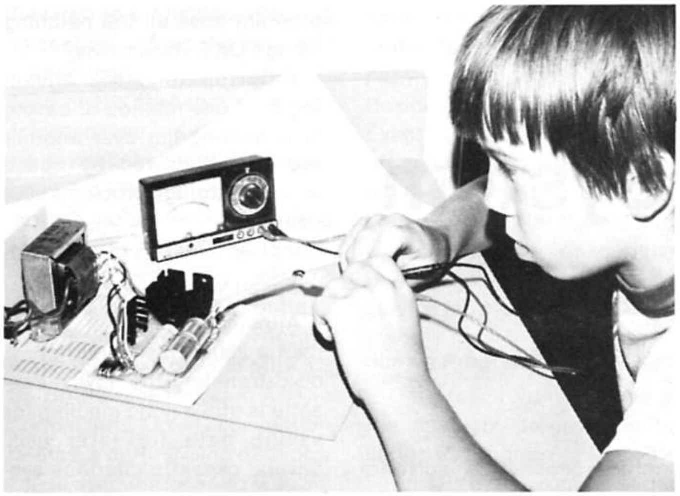
All parts were checked off by Mark, which helped him become familiar with the parts and learn their use. As he checked them, I took a few moments to explain the function of the various parts. Everything was there. Some kits don't always include all items; but North Star has it all together.

Mark installed the 47 IC sockets and soldered them in place. He had soldered a few times before so he was familiar with the correct circuit-board soldering procedure. He had a few problems with bridges, but a little Solder-Wick removed them. I was pleased to see a very professional soldermasked board; properly soldermasking a board helps to eliminate problems.

The eight resistors and 40



What you see is what you get. The kit comes complete for the North Star Disk System. The Shugart disk drive (back right) comes complete and assembled.



WOW! Five volts. After the power supply was completed, the connector plug was checked for correct voltages. All OK.

capacitors were then soldered to the board, and the crystal, 5 volt regulator and heat-sink hardware followed. It was now necessary to solder a 34 pin cable connector to the board. The MDS Controller board was plugged into the computer. Holding his breath, Mark connected the meter, which read + 5 volts. So far so good.

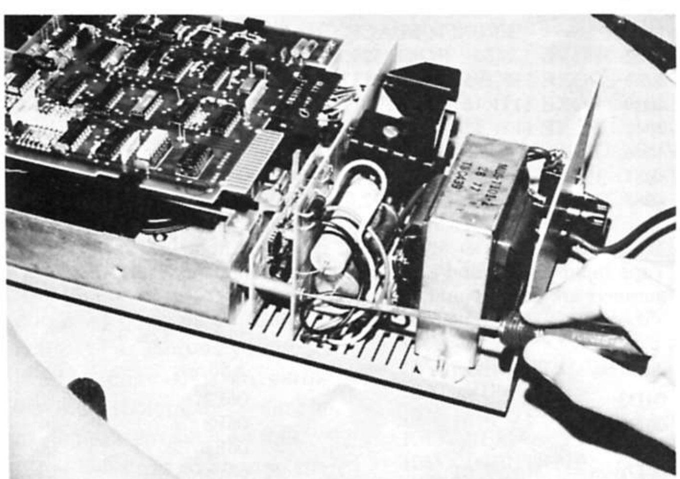
IC Installation

Mark watched while I demonstrated the correct way to install the ICs in the sockets. I made a quick check to make sure he had them in the correct location. The manual then gave two detailed pages of instruction for waveforms on a scope. Since I did not have a scope available we skipped this step.

Power Board Assembly

The disk drive can receive power three ways: (1) From +5 and +12 volts from an existing power supply; (2) power PC board to regulate power from an existing unregulated power supply; (3) North Star power-supply option (MDS-PS).

Since I knew we would be using the North Star with two different computers from one time to another, I had purchased the North Star power-supply option. Mark mounted the transformer in the cabinet and hooked up the wires, switch and fuse to complete the power supply. Ready to test the power supply for +5 and +12 volts at the power plug, Mark hooked up the meter and checked for the proper voltages. To our satisfaction, they were OK.



Disk drive assembly. The power supply is assembled to the disk drive assembly with two spacers and screws on each side. The unit is then connected to the case.

Look Dad, I did it! A very proud young man. If he can build the North Star System, you can. Let's go.

The last thing to do was to make two trace cuts on the MDS controller board and install two jumper wires. Done!

Final Check

The real test was drawing near. Mark installed the MDS controller board in the computer and hooked up the cables. With the power switch and computer on and the disk in the disk drive, Mark typed EX E900 and hit return. As he did that I explained that an asterisk on the screen signaled that everything was OK. The next command was GO BASIC. Mark did that and BASIC was loaded in 2 seconds. READY appeared on the screen and we were ready to program.

Up And Running

Mark and I input a small program to make sure everything was OK. It was. We sat at the computer for over three hours inputting programs and running them. It was getting late, so we stopped and decided to input some more programs during the next few days.

Summary

Total construction time for the project was 4 hours and 20 minutes. You could probably complete it in less time if you have experience building kits. Mark took his time building the kit, but the time spent paid off because the system worked the first time.

While Mark was running a few programs, I looked over the manuals. North Star BASIC is an extended version and has numerous functions. It also has an edit function to correct errors; it is a joy to use.

The OEM manual gives complete and detailed description of the disk drive and complete schematics. The North Star Disk Operating Systems Manual features complete instructions and operations for the DOS. It contains descriptions on creating files, types of files, deleting files, jump routines, read and write and many more procedures that are available for use. All the manuals are written so you can understand them. Maybe some other manufacturers will take a lesson from North Star. ■

A Simple Mailing System

a money-making time-saver

Stephen Gibson
PO Box 38386
Los Angeles CA 90038

One of the first tasks a small businessman wants his new computer to do is handle the company mailing list. A review of the many programs available reveals a big problem: Unless you have a disk storage system, you are forever condemned to load all those names and addresses via the DATA statement.

To read out the list or print labels, the data is usually read into a set of variables, then formatted to fit your particular hard-copy printer. If you fill your memory or want separate lists, you have to write a whole new program. To update, you are forced to list the program to find where the last DATA statement ended, then change the read routine.

All this nonsense takes valuable time and makes you a slave to the machine. It would be easier to write the program only once, and simply change the lists. Here are two ways to do it: (1) the cassette method and (2) the bare-bones method. At least one is bound to work for you.

Sneaky Software Secrets Revealed

The problem is not how to structure the ideal list program in BASIC, but how to save the names and addresses in a language that doesn't know how to save variables. The main program should have to be saved only once.

Surprisingly enough, a number of rather clever techniques have been developed to solve this problem. One method breaks down the name, a string variable, and feeds it to tape as a series of OUT statements. Another method uses the tape interface hard-wired in parallel with the terminal I/O. Still another chooses a software method by patching BASIC's terminal I/O over to the tape interface port, and then outputs the list via PRINT statements as though it were the terminal . . . very clever, because the program does all the patching using POKE statements.

Unfortunately, the advantages of one method of saving your mailing list over another are overshadowed by speed and tape storage problems with your unit.

Is Speed Your Thing?

Although somewhat slower than a 250K bit/second floppy-disk transfer, the lowly cassette is still a good medium for saving data for later use. Several cassette interface systems are available. They differ widely with respect to speed.

I picked the Tarbell high-speed interface and coupled it to the Data Duffer (see *Kilobaud*, March 1978, "Hear It and

The entire system here is an Imsai 8080 with 24K of memory, ADM-3 terminal, Data Duffer, Teletype . . . and one efficient secretary.

Add these lines to the program to let the computer tell you when to abbreviate.

```
12   A = 20 :REM WIDTH OF TTY LABEL
1050  IF LEN(NA$(N)) > A THEN GOSUB 5000 : GOTO 1040
1060  IF LEN(CO$(N)) > A THEN GOSUB 5000 : GOTO 1055
1070  IF LEN(AD$(N)) > A THEN GOSUB 5000 : GOTO 1065
1080  IF LEN(CS$(N)) > A THEN GOSUB 5000 : GOTO 1075
1090  IF LEN(ZP$(N)) > A THEN GOSUB 5000 : GOTO 1085
5000  REM
5005  REM LINE LENGTH ERROR
5010  REM
5015  PRINT:PRINT"LINE TOO LONG!!":PRINT:RETURN
```

Add these lines to run the program with Mits 3.2 12K BASIC

```
3020  POKE 1776,110 : POKE 1778,32 : POKE 1784,1
3300  POKE 1776,0 : POKE 1778,128 : POKE 1784,1
4015  POKE 1787,110 : POKE 1789,16 : POKE 1794,111
4020  POKE 1778,0 : POKE 1784,255
4090  POKE 1787,0 : POKE 1789,1 : POKE 1794,1
4095  POKE 1778,128 : POKE 1784,1
```

Add these lines to modify Mits 3.2 8K BASIC to recognize leading spaces.

```
13   SP$ = " " :REM A SPACE CHARACTER
4022  POKE 528,54 : POKE 529,32 : POKE 530,35
4023  POKE 531,195 : POKE 532,224 : POKE 533,7
4039  POKE 1171,16 : POKE 1172,2
4041  POKE 1171,224 : POKE 1172,7
4046  IF B$ = (E$ + SP$) THEN 4090
4087  POKE 528,0 : POKE 529,0 : POKE 530,0
4088  POKE 531,0 : POKE 532,0 : POKE 533,0
```

Make these patches to Mits BASIC if you get hung up in the Tape Input routine and need to return to command level. All numbers are hexadecimal.

8K 3.2		**12K 3.2**	
Address	Byte	Address	Byte
04D3	80	06F2	80
04D9	01	06F8	01
04DC	00	06FB	00
04DE	01	06FD	01
04E3	01	0702	01

Fig. 1. Mits BASIC patches.

See It!") as a reliable way to use cassettes without the hassle of a seemingly endless wait for a load or the fear that data was lost because a switch was off or a knob twisted the wrong way. The Tarbell manual suggests a variable-saving method in which the terminal I/O is software patched to the cassette I/O for a transfer. The routines in the mailing-list program make these patches to Mits 3.2 8K BASIC (see Program A). The normal Mits TTY I/O convention of status port "0" and data port "1" is used. Patches to Mits 3.2 12K BASIC are also listed in Fig. 1. If you don't have Mits BASIC or a Tarbell, there's still hope; you can use the bare-bones method described later.

Hard-Copy Hassles

Registration is the key ingredient for alignment of the labels on your printer. A sprocketed feed mechanism is almost a necessity. Of course, you can simply cut your labels out with a large paper-cutter, but the peel-off-type labels are more convenient and better looking. You need the sprocket feed to make them work properly. You might even consider custom labels with fancy artwork or the company logo.

I had quite a time finding off-the-shelf labels for my old sprocket-fed Teletype. Almost everyone sells ready-made forms for larger printers. There are a few companies that specialize in stock or custom labels from camera-ready artwork (see accompanying "Sources for More Information").

If you do start with a Teletype, by all means change the ribbon! Use a carbon ribbon rather than the stock cloth one —the printing looks so much better. Unique type fonts are also available for the Teletype. Even the Teletype can be made to look as good as an IBM Selectric . . . as long as you don't mind all caps—not an earth-shaking problem for a simple mailing system such as this.

You will have to change the platen if your Teletype is a friction-feed model. The modification to your machine is simple and inexpensive. I'm not advocating the Teletype as the ideal printer for this system; my company just happens to have one. Besides being slow, it's noisy! Eventually, I had to stick ours off in a room by itself to drown out the clatter. The advantages, of course, are that the machine is reliable and inexpensive. Used machines abound, and service is readily available.

Simple Program Does It All

Only four routines make up the cassette program. In the listing in Program A, lines 1 to 50 initialize the program. A generous 10,000 bytes are cleared away based on an average line length of 20 characters, with 5 lines given to each company and a list size of 100 companies. The variables S and L represent the maximum size of the list and the current list size, respectively. The subscripted variables in line 25 are dimensioned to the size of the list. Of course, you can set this value higher for a larger list if your memory capacity will permit it. The command level routine prints suitable prompts for those unfamiliar with the program. A branch is made at line 155 based on the value of C.

To enter names at line 1000, the list counter L is incremented by 1 and a test is made to see if the list size is greater than 100 names. It might be later on, so we must check it out. If so, the list counter is decremented back to 100 and a return is made to the command routine. In line 1030, a message indicates that the number symbol (#) can be used to exit the routine. A FOR/NEXT loop inputs the names and addresses into the subscripted variables.

You might wish to make the prompts different for your version. Instead of "ZIP . . . ," for instance, you might want the program to print "COUNTRY . . . ," if you mail overseas. Or you could eliminate "ZIP" (ZP$) altogether and squeeze it into the CITY/STATE line.

If # is typed in line 1045, a branch is made and the list

```
1 REM ------------------------------------------------------------
2 REM            **** MAILING LIST ****
3 REM ------------------------------------------------------------
4 REM BY STEPHEN GIBSON 1/10/77
5 REM RUNS ONLY ON MITS 3.2 8K BASIC
6 REM AND TARBELL CASSETTE INTERFACE
7 REM ------------------------------------------------------------
8 REM INITIALIZE
9 REM ------------------------------------------------------------
10 CLEAR 10000 :REM CLEAR SPACE FOR LIST
15 S = 100 :REM MAXIMUM LIST SIZE
20 L = 0 :REM CURRENT LIST SIZE
25 DIM NA$(S),CO$(S),AD$(S),CS$(S),ZP$(S)
30 E$ = "#"  :REM END OF LIST CHARACTER
35 OUT 1,26  :REM CLEARS SCREEN
40 PRINTTAB(20);"*** THIS IS MAILING LIST ***"
50 PRINT
100 REM ----------------------------------------------------------
105 REM COMMAND LEVEL ROUTINE
110 REM ----------------------------------------------------------
115 PRINT"PLEASE ENTER YOUR COMMAND:":PRINT
120 PRINT"ENTER NAMES INTO LIST   = 1"
125 PRINT"PRINT-OUT OF LIST       = 2"
130 PRINT"STORE LIST ON TAPE      = 3"
135 PRINT"READ LIST FROM TAPE     = 4"
140 PRINT
145 INPUT"COMMAND";C
150 IF C > 4 THEN 115
155 ON INT(C) GOTO 1000, 2000, 3000, 4000
1000 REM ---------------------------------------------------------
1005 REM ENTER NAMES ROUTINE
1010 REM ---------------------------------------------------------
1015 L=L+1
1020 IF L > 100 THEN 1400
1025 PRINT"IF YOU WISH TO EXIT THIS ROUTINE . . . "
1030 PRINT"TYPE ONE OF THESE '#',THEN 'RETURN'."
1035 FOR N = L TO 100 :PRINT:PRINT"NUMBER ";N:PRINT
1040 INPUT"NAME :            ";NA$(N)
1045 IF NA$(N) = "#" THEN 1300
1047 IF NA$(N) = "\" THEN N=N-2 :GOTO 1100
1050 REM
1055 INPUT"COMPANY :         ";CO$(N)
1060 REM
1065 INPUT"ADDRESS :         ";AD$(N)
1070 REM
1075 INPUT"CITY & STATE :  ";CS$(N)
1080 REM
1085 INPUT"ZIP :             ";ZP$(N)
1100 NEXT
1200 L = 100 : GOTO 1500
1300 L = N - 1 : GOTO 1600
1400 L = L - 1
1500 PRINT:PRINT"THE LIST IS FULL.":PRINT
```

```
1600 PRINT:PRINT"YOU HAVE ";L;"NAMES ON THIS LIST."
1700 GOTO 100
2000 REM ----------------------------------------------------------------------
2005 REM  PRINT-OUT ROUTINE
2010 REM ----------------------------------------------------------------------
2015 REM  PRINT
2020 PRINT"1) LINE UP LABELS IN PRINTER."
2025 PRINT
2030 PRINT"2) TURN ON PRINTER."
2035 PRINT
2040 PRINT"3) TYPE ANY LETTER, THEN 'RETURN'."
2045 PRINT:INPUT"WAITING . . . ";W$
2050 FOR X = 1 TO L STEP 3
2055 Y = X + 1 : Z = X + 2
2060 PRINT TAB(0) ; NA$(X) ; TAB(25) ; NA$(Y) ; TAB(51) ; NA$(Z)
2065 PRINT TAB(0) ; CO$(X) ; TAB(25) ; CO$(Y) ; TAB(51) ; CO$(Z)
2070 PRINT TAB(0) ; AD$(X) ; TAB(25) ; AD$(Y) ; TAB(51) ; AD$(Z)
2075 PRINT TAB(0) ;  CS$(X) ; TAB(25) ;  CS$(Y) ; TAB(51) ;  CS$(Z)
2080 PRINT TAB(0) ;  ZP$(X) ; TAB(25) ;  ZP$(Y) ; TAB(51) ;  ZP$(Z)
2085 PRINT:PRINT
2090 NEXT
2095 GOTO 100
3000 REM ----------------------------------------------------------------------
3005 REM  STORE ON TAPE ROUTINE
3010 REM ----------------------------------------------------------------------
3011 PRINT:PRINT"1) PLACE NEW CASSETTE IN RECORDER."
3012 PRINT:PRINT"2) PUT IN RECORD MODE AND ZERO COUNTER."
3013 PRINT:PRINT"3) WAIT A FEW SECONDS TO ALLOW A LEADER."
3014 PRINT:INPUT"4) TYPE ANY LETTER, THEN 'RETURN'.";W$
3015 S$ = CHR$(195) + CHR$(230)
3020 POKE 1233,110 : POKE 1235,32 : POKE 1241,111
3025 FOR N = 1 TO L
3030 D$(1) = NA$(N)
3035 D$(2) = CO$(N)
3040 D$(3) = AD$(N)
3045 D$(4) = CS$(N)
3050 D$(5) = ZP$(N)
3055 FOR J = 1 TO 5
3060 FOR K = 1 TO 100 : NEXT K
3065 B$ = S$ + D$(J)
3070 PRINT B$
3075 NEXT J
3080 NEXT N
3085 FOR T = 1 TO 3
3090 B$ = S$ + E$
3095 FOR K = 1 TO 100 : NEXT K
3100 PRINT B$
3200 NEXT T
3300 POKE 1233,0 : POKE 1235,128 : POKE 1241,1
3400 GOTO 100
4000 REM ----------------------------------------------------------------------
4005 REM  READ FROM TAPE ROUTINE
4010 REM ----------------------------------------------------------------------
4011 PRINT:PRINT"1) PLACE CASSETTE IN RECORDER."
4012 PRINT:PRINT"2) SET COUNTER AND PUSH PLAY."
4013 PRINT:PRINT"3) ALLOW TIME FOR LEADER."
4014 PRINT:INPUT"4) TYPE ANY LETTER, THEN 'RETURN'.";W$
4015 POKE 1244,110 : POKE 1246,16 : POKE 1251,111
4020 POKE 1235,0 : POKE 1241,255
4025 FOR N = 1 TO 101
4030 FOR J = 1 TO 5
4035 OUT 110,16
4040 INPUT B$
4045 IF B$ = E$ THEN 4090
4050 D$(J) = B$
4055 NEXT J
4060 NA$(N) = D$(1)
4065 CO$(N) = D$(2)
4070 AD$(N) = D$(3)
4075 CS$(N) = D$(4)
4080 ZP$(N) = D$(5)
4085 NEXT N
4090 POKE 1244,0 : POKE 1246,1 : POKE 1251,1
4095 POKE 1235,128 : POKE 1241,1
4100 L = N - 1
4200 PRINT:PRINT"THIS LIST HAS ";L;" NAMES ON IT.":PRINT
4300 GOTO 100
```

Program A. Program listing for A Simple Mailing System. Here are the routines you need to patch Mits 8K 3.2 BASIC to load or save your mailing list using the Tarbell high-speed cassette interface.

counter L is decremented by one (1) and a return is made to the command routine. Sometimes I make mistakes when entering a name (my secretary never does). I find it convenient to be able to type a character that tells the program to go back one name and start over. Line 1047 does it all. I chose a backslash, but you should feel free to choose your own character to personalize this program. You could insert this line after every input if you'd rather check your work a line at a time.

Another useful addition is to print a space, for example, where the name goes in the event you have a company name, but no one individual to mail to. A space is a logical entry. Don't try it unless you add the appropriate lines from Fig. 1 because Mits BASIC ignores leading spaces. The listed POKEs change the input routine to add a space if a carriage return is received. I found it convenient to print the current list size in line 1600 before exiting this routine.

The printout routine must be tailored to your particular printer. The program format given is for a standard Teletype using peel-off labels spaced three across. Lines 2020 to 2045 give instructions. The variable W$ is only a buffer to wait until you are ready to print. Extra PRINT statements in line 2085 advance the form to the next set of labels. To print your labels three at a time for the popular machine-gun mailings, simply substitute the lines in Fig. 2.

The store (on tape) routine at line 3000 begins the really useful aspects of this program. It is here that the names and addresses only are fed to tape. Instructions are given in lines 3011 to 3014. W$ is still only a wait buffer. S$ is set to the value of the Tarbell start and sync bytes. POKEs to Mits BASIC are then made in line 3020 to shift the terminal I/O to the cassette I/O port. The names and addresses are placed in a D$ buffer, then output with the start and sync bytes as B$ via PRINT statements.

```
Instead of this format . . .

John Craig             Wayne Green            Stephen Gibson
Editor                 Publisher              Famous author
Kilobaud Magazine      Kilobaud Magazine      PO Box 38386
Peterborough NH        Peterborough NH        Los Angeles CA
03458                  03458                  90038

You might want this . . .

John Craig             John Craig             John Craig
Editor                 Editor                 Editor
Kilobaud Magazine      Kilobaud Magazine      Kilobaud Magazine
Peterborough NH        Peterborough NH        Peterborough NH
03458                  03458                  03458

Wayne Green            Wayne Green            Wayne Green
Publisher              Publisher              Publisher
Kilobaud Magazine      Kilobaud Magazine      Kilobaud Magazine
Peterborough NH        Peterborough NH        Peterborough NH
03458                  03458                  03458

Then substitute these lines . . .

2050  REM  3-UP FORMAT
2055  FOR N = 1 TO L
2060  PRINT TAB(0);NA$(N);TAB(25);NA$(N);TAB(51);NA$(N)
2065  PRINT TAB(0);CO$(N);TAB(25);CO$(N);TAB(51);CO$(N)
2070  PRINT TAB(0);AD$(N);TAB(25);AD$(N);TAB(51);AD$(N)
2075  PRINT TAB(0); CS$(N);TAB(25); CS$(N);TAB(51); CS$(N)
2080  PRINT TAB(0); ZP$(N);TAB(25); ZP$(N);TAB(51); ZP$(N)
```

Fig. 2. Instead of this format . . .

The delay loop in line 3060 bears some explanation. When data is brought back into the program, allow time for BASIC to reinsert the data into the appropriate subscripted variables by implementing a delay during the output sequence. You could, perhaps, shorten the delay, but you might lose some of your data. A value of 100 for T allows plenty of safety.

The End of List character, E$, must also be output. The computer will look for this character when the list is played back into the machine to set the list counter. This particular arrangement allows lists of varying size and the addition of more names to a short list.

Beginning on line 3090, E$ is linked to the start and sync bytes and output three times. Why three; isn't once enough? That's right. But suppose you had a dropout on the tape. It does happen on old cassettes, particularly cheap ones. Even if you use top-notch cassettes, you may still lose a byte because your recorder's slow AGC attack time may turn the beginning of a byte to garbage. I proved it writing this program.

The computer missed the E$ on playback. It just sat there waiting. It was very annoying . . . especially because the program had POKEd the I/O away from my terminal to the cassette interface. I had no way to talk to my machine except via the system monitor and the front panel to patch things up between my computer and its program. The pandemic Murphy's Law says you won't need to use the patches I made if I list them in Fig. 1. I output the E$ three times, rather than once, and beat old Edsel Murphy by even a New York second! (That's easy for me to say, you say.) The routine ends by POKEing BASIC back to normal I/O and jumping to the command routine.

The tape input routine is very similar. Instructions are given in lines 4011 to 4014. The I/O is POKEd to the cassette port just as before, and data is input by another FOR/NEXT loop. It is useful to print out the size of the list after the I/O is POKEd back because not all lists will be set at the maximum size. You will then be able to add to the current list by using the input routine. Then save the whole thing as a full list.

The Bare-Bones Method

Suppose you have a computer and a Teletype, but neither speaks Mits BASIC nor recognizes Tarbell format. If your Teletype has a paper-tape punch (most do), you can still benefit from this system.

Start by making those nifty mods to the Teletype, especially the ribbon. Then enter the program in Program B. The variables are the same as the cassette program, but the prompts are different and the save and read routines are left out.

Next, run the program and enter the names and addresses. When you print the list, simply turn on the paper-tape punch at the same time. You will have an exact copy of the printout, as well as a set of labels, on paper tape. You can then reprint the list by using the Teletype in the local mode and reading off the paper tape. Turn on the punch again while printing if you need a spare copy of your list. Use a separate punch if you have one.

```
1   REM  ------------------------------------------------------------
2   REM        **** BARE BONES MAILING LIST ****
3   REM  ------------------------------------------------------------
4   REM  BY STEPHEN GIBSON 12/11/76
5   REM  RUNS ON ASR-33 TTY OR SIMILAR
6   REM  PRINTER WITH PAPER TAPE PUNCH
7   REM  ------------------------------------------------------------
8   REM  INITIALIZE
9   REM  ------------------------------------------------------------
10  CLEAR 10000 :REM CLEAR SPACE FOR LIST
15  S= 100 :REM MAXIMUM LIST SIZE
20  L = 0 :REM CURRENT LIST SIZE
25  DIM NA$(S),CO$(S),AD$(S),CS$(S),ZP$(S)
30  E$ = "#" :REM END OF LIST CHARACTER
35  OUT 1,26 :REM CLEARS SCREEN
40  PRINTTAB(20);"*** THIS IS MAILING LIST ***"
50  PRINT
100  REM  -----------------------------------------------------------
105  REM  COMMAND LEVEL ROUTINE
110  REM  -----------------------------------------------------------
115  PRINT"PLEASE ENTER YOUR COMMAND:":PRINT
120  PRINT"ENTER NAMES INTO LIST    = 1"
125  PRINT"PRINT-OUT OF LIST        = 2"
140  PRINT
```

```
145  INPUT"COMMAND";C
150  IF C > 2 THEN 115
155  ON INT(C) GOTO 1000, 2000
1000 REM  -----------------------------------------------------------------------------------
1005 REM  ENTER NAMES ROUTINE
1010 REM  -----------------------------------------------------------------------------------
1015 L=L+1
1020 IF L > 100 THEN 1400
1025 PRINT"IF YOU WISH TO EXIT THIS ROUTINE . . . "
1030 PRINT"TYPE ONE OF THESE '#',THEN 'RETURN'."
1035 FOR N = L TO 100 :PRINT:PRINT"NUMBER ";N:PRINT
1040 INPUT"NAME :            ";NA$(N)
1045 IF NA$(N) = "#" THEN 1300
1047 IF NA$(N) = "\" THEN N=N-2 :GOTO 1100
1050 REM
1055 INPUT"COMPANY :         ";CO$(N)
1060 REM
1065 INPUT"ADDRESS :         ";AD$(N)
1070 REM
1075 INPUT"CITY & STATE :  ";CS$(N)
1080 REM
1085 INPUT"ZIP :             ";ZP$(N)
1100 NEXT
1200 L = 100 : GOTO 1500
1300 L = N - 1 : GOTO 1600
1400 L = L - 1
1500 PRINT:PRINT"THE LIST IS FULL.":PRINT
1600 PRINT:PRINT"YOU HAVE ";L;" NAMES ON THIS LIST."
1700 GOTO 100
2000 REM  -----------------------------------------------------------------------------------
2005 REM  PRINT-OUT ROUTINE
2010 REM  -----------------------------------------------------------------------------------
2015 PRINT
2020 PRINT"1) MAKE PAPER TAPE LEADER IN 'LOCAL' MODE."
2025 PRINT
2030 PRINT"2) SWITCH PRINTER TO 'LINE' AND LINE UP LABELS."
2035 PRINT
2040 PRINT"3) TYPE ANY LETTER, THEN 'RETURN'."
2045 PRINT:INPUT"WAITING . . . ";W$
2050 FOR X = 1 TO L STEP 3
2055 Y = X + 1 : Z =X + 2
2060 PRINT TAB(0) ; NA$(X) ; TAB(25) ; NA$(Y) ; TAB(51) ; NA$(Z)
2065 PRINT TAB(0) ; CO$(X) ; TAB(25) ; CO$(Y) ; TAB(51) ; CO$(Z)
2070 PRINT TAB(0) ; AD$(X) ; TAB(25) ; AD$(Y) ; TAB(51) ; AD$(Z)
2075 PRINT TAB(0) ;  CS$(X) ; TAB(25) ;  CS$(Y) ; TAB(51) ;  CS$(Z)
2080 PRINT TAB(0) ;  ZP$(X) ; TAB(25) ;  ZP$(Y) ; TAB(51) ;  ZP$(Z)
2085 PRINT:PRINT
2090 NEXT
2095 GOTO 100
```

Program B. Listing for the bare-bones version of the program. The format is set for a Teletype. Simple adjustments can be made to fit other printers. A paper-tape punch is used to save the list. The Teletype is run in local mode to print additional lists.

This particular method is inexpensive and does not take any time at all to load or make because the paper-tape copy is punched as you print the list! How easy can something be?

If It Works . . . Modify It!

Suppose your names are longer than your labels. When do you abbreviate? Adding the appropriate lines from Fig. 1 allows the computer to count the number of input characters. The LEN function, if you have it (Mits does), can test against the size of your line. If the test is valid, GOSUB to an error message. Further modifications include another module to read a whole letter from cassette using the POKEs given. Then print a personalized copy to each customer on the list.

You can now consider sentence structures like "and in closing, 'Mr. Jones,' we'd like to offer . . . ," just as the big mail-order operations do it! Still another useful modification is a cassette tape directory of your lists . . . a good idea when you get up to a thousand. An excellent example of this method appeared in *73 Magazine* ("The Soft Art of Programming," Parts 1-3, Oct-Dec 1976, by Rich Didday) and was reprinted in *The New Hobby Computers*, 73 Inc., 1977.

We don't have to confine our list to names and addresses. Adding a few more variables in the program allows the luxury of obtaining other important data from our list, such as types of merchandise each customer wants or has ordered. You might choose to save important dates for each customer—write a simple routine to search the current list and pop out names that need collection letters, birthday greetings or warranty follow-up letters. The personalized form letter, mentioned before, could be printed just for those on the list who need it. All you need do is add to the routines given.

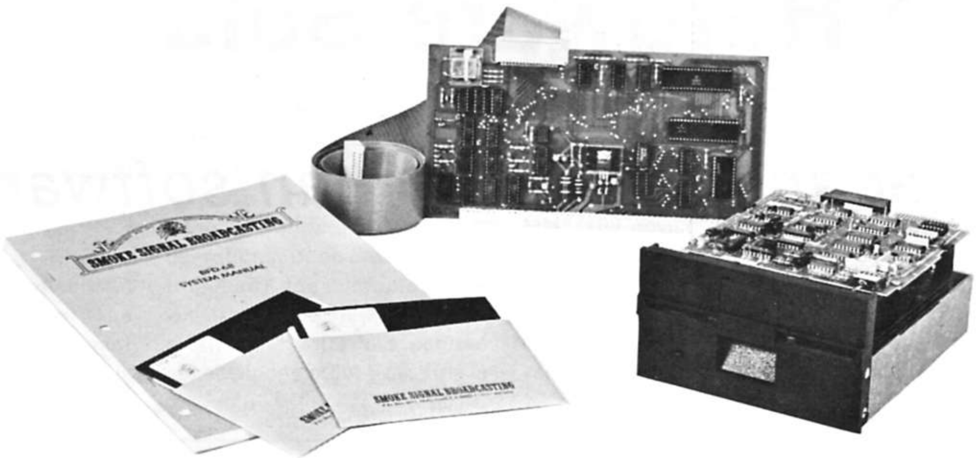
Perhaps you can begin to see that what started as a simple system could easily be expanded into a first-class data base for your business. You can start with the program given and upgrade from there, even to disk. You lose nothing by starting now with just the list. In fact, you may gain in the long run because you will be able to tailor the program to your own needs. The really important procedures will be yours, thereby ending forever that locked-in feeling you get with someone else's software.

If you know that feeling or need an upward compatible mailing program for your business, you should get this program up and running and begin to save time and money *now* while planning for the future. ■

Sources For More Information

High-speed cassette interface.	Tarbell Electronics 20620 S. Leapwood Ave. Suite P Carson CA 90746 (213) 538-4251
Teletype labels and ready-made forms for printers.	Uarco Incorporated 2600 Wilshire Bl. Suite 408 Los Angeles CA 90057 (213) 380-2595
Custom labels for any printer.	Avery Label Company 777 E Foothill Bl. Azusa CA 91704 (213) 969-3311
Teletype sprocket feed kits and special type fonts. Also carbon ribbons for Teletype.	TTS 2928 Nebraska Ave. Santa Monica CA 90404 (213) 829-2611

THE SSB $150 FLOPPY DISCOUNT



Affordable

The tribe at Smoke Signal Broadcasting took our BFD-68 disk system and scalped the price, but not the features to create the ABFD-68 (Affordable Basic Floppy Disk). We appreciate the fact that the computer hobbyist gave us our start and we haven't forgotten you.

$649 Assembled

Compare Price. Our SS-50 bus compatible disk system is $150 less than the assembled price of the leading S-100 disk system. And you can at least double that savings when you buy one of the computers manufactured by MSI or SWTPC that use the superior 6800 microprocessor.

Programmable

The BFD-68 is well known for its fine software. The system comes with the best disk operating system available and we offer a multitude of other compatible software products. These include a BASIC interpreter with disk file handling capability. By the way, our DOS now easily handles true random access files as well as sequential. Also, we have a super fast BASIC compiler for business applications. In addition, a Text Editor, 2 Assemblers, a Trace Disassembler useful for program debugging and an Object to Source Code Generator are all stock items available for immediate delivery. A word processor will be available very soon.

Reliable

We delivered our first mini-floppy disk system a year ago — 6 months ahead of any other 6800 based mini system. Thus, we've had twice the experience in building reliability into the system. Our NEW disk controller was designed using all we have learned in the past year about system reliability.

The ABFD-68 contains all the built in reliability of our regular BFD-68 plus you save money by supplying your own cabinet and power supply for the disk.

Available

We've shipped literally tons of our BFD-68 disk system in the past year and have learned to keep our production up with demand. Give us a call and chances are we'll be able to ship you the new ABFD-68 from stock and charge it to your Master Charge or Visa card. Better yet, ask us for the name of the computer store nearest you that carries our complete line of computer products.

SMOKE SIGNAL BROADCASTING

P.O. Box 2017, Hollywood, CA 90028 • (213) 462-5652

Dr. Adam Osborne
Osborne & Associates, Inc.
PO Box 2036
Berkeley CA 94702

Number Crunching: Two Hardware Solutions

faster and smoother than software

People attempting to use microprocessors in scientific applications are probably the first to discover that microprocessors do indeed have limitations. A microprocessor's ability to execute instructions in microseconds may, on the surface, sound very impressive, and it is—until you try to handle trigonometric functions, logarithms, exponentiation or even multidigit multiplication and division.

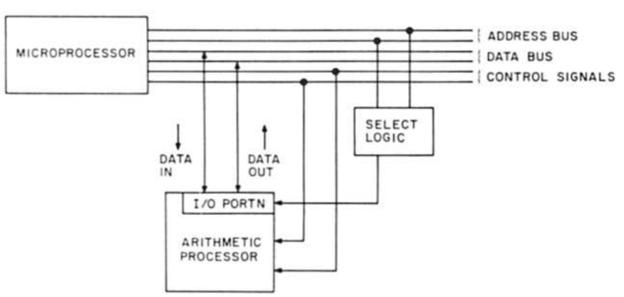


Fig. 1. An arithmetic processor in a microcomputer system.

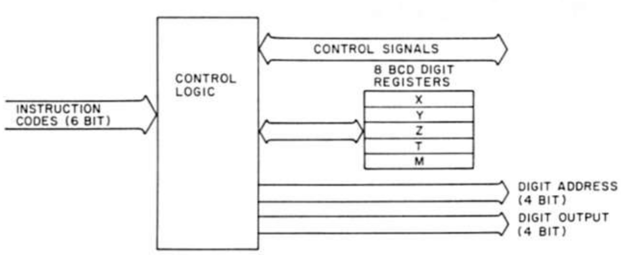


Fig. 2. MM57109 arithmetic processor functional logic.

Trigonometric and logarithmic functions are generally referred to as "transcendental" functions. Writing a microcomputer program to handle transcendental functions is far more difficult than the most complex payroll system could ever be. In fact, designing a program that will generate truly accurate transcendental functions is a formidable task. The problem with these functions is that over limited ranges they change rapidly. Programs that generate transcendental functions must generate very accurate answers, particularly in the fast-moving range, because on rare occasions you will want to subtract almost identical values—and a small difference between two large, erroneous numbers may be completely wrong.

Therefore, when examining arithmetic processors, you must look at the accuracy of results in the fast-moving numeric range. If you are accustomed to evaluating chips simply on the basis of cycle times and programmable options, you now have an important new consideration—the method used to generate results.

Two arithmetic processors will soon be available: the MM57109 from National Semiconductor and the AM9511 from Advanced Micro Devices. About the only thing these two devices have in common is that they both perform approximately the same transcendental functions, and each is treated as a support device within a microcomputer system.

Suppose, for example, you want to compute the natural logarithm of a number. You will transmit the number, as data, to an arithmetic processor, addressing it as an I/O port. At some later time you will read back the answer, as data being input from an I/O port. This use of an arithmetic processor is illustrated conceptually in Fig. 1.

The primary difference between the MM57109 and the AM9511 is that National Semiconductor's MM57109 is a calculator chip; it looks nothing

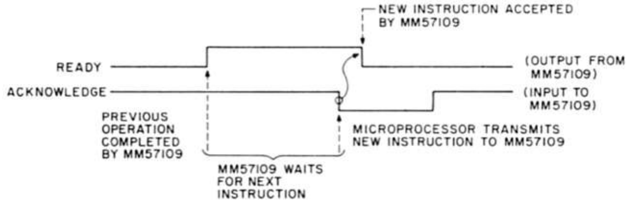


Fig. 3.

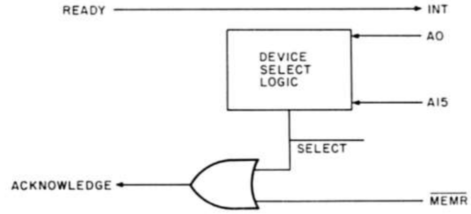


Fig. 4.

like the typical microprocessor support device. The AM9511, in contrast, is immediately recognizable to any experienced microcomputer user as a typical microprocessor support device.

Let's look at each part in turn. The discussion that follows will give you some idea of part capabilities; however, detailed operating procedures are not provided.

MM57109

Fig. 2 illustrates the general logic organization of the MM57109. The most important characteristic of this part is that it operates on binary-coded decimal (BCD) numbers up to eight digits long. Numbers may be handled in fixed-point or floating-point format. A fixed-point number is eight digits long, with a decimal point located at any digit boundary. Thus, numbers in the range 99999999 through .00000001 may be represented.

Floating-point numbers have the form:

$$(\pm 0.XXXXXXXX)ExP(\pm YY)$$

X and Y represent any decimal digits. Thus, any number in the range $1 \times 10^{+99}$ through 1×10^{-99} can be represented, with eight digits of accuracy.

As you might expect, you must operate the MM57109 by transmitting data and commands to it. Results are received as data. Commands are summarized in Table 1. Note that the MM57109 is not a fast device. Execution times are shown based on a ten-microsecond microcycle, the recommended maximum rate for this device. It takes at least four milliseconds to enter a single eight-digit number (in fixed- or floating-point notation), while trigonometric functions may take almost a second to resolve.

In order to cope with these relatively slow times, all data communications between the MM57109 and a microprocessor use request/acknowledge handshaking control signal protocol. Upon completing any operation, the MM57109 outputs a ready signal true. Normally the microprocessor will hold an acknowledge input false to suppress any new operations. Upon detecting the true ready, the microprocessor will transmit a new command to the MM57109 and set the acknowledge input true. This is illustrated in Fig. 3.

This handshaking scheme readily lends itself to almost any microprocessor; the ready "true" signal can be used to request an interrupt, while the acknowledge can be tied directly to a combined MM57109 device-select and write-control signal. For 8080A signals, this is illustrated in Fig. 4.

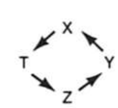
The method of transmitting control commands to an MM57109 device differs markedly from the standard method used within microcomputer systems. The standard method (which is used by the AM9511) takes the device-select logic output to a select pin, then has a control/data discriminator that usually constitutes part of the device address. Memory-read and memory-write control signals then become simple control strobes that accompany an address-activated select logic. Fig. 5 illustrates this.

There are three ways you can enter data to the MM57109; in each case the register stack is pushed and data is written into the X register (see Fig. 6).

The first data entry method is approximately equivalent to calculator-keyboard entry; separate commands identify the

*Table 1. MM57109 instruction description table (*indicates two-word instruction).*

MNEMONIC*	EXECUTION TIME (MICROCYCLES) (AVERAGE)	EXECUTION TIME (MICROCYCLES) (WORST-CASE VALUES)	DESCRIPTION
0		238	Mantissa or exponent digits. On first digit (d) the following occurs: Z→T
1		238	Y→Z
2		238	X→Y
3		238	d→X
4		238	See description of number entry on page 11.
5		238	
6		238	
7		238	
8		238	
9		238	
DP		152	Digits that follow will be mantissa fraction.
EE		151	Digits that follow will be exponent.
CS		166	Change sign of exponent or mantissa. Xm = X mantissa Xe = X exponent CS causes −Xm→Xm or −Xe→Xe depending on whether or not an EE instruction was executed after last number entry initiation.
PI		1312	3.1415927→X, stack not pushed.
EN		552	Terminates digit entry and pushes the stack. The argument entered will be in X and Y. Z→T Y→Z X→Y
NOP		122	Do nothing instruction that will terminate digit entry.
HALT		134	External hardware detects HALT op code and generates HOLD = 1. Processor waits for HOLD = 0 before continuing. HALT acts as a NOP and may be inserted between digit entry instructions since it does not terminate digit entry.
ROLL		905	Roll Stack.



POP		448	Pop Stack. Y→X Z→Y T→Z O→T
XEY		652	Exchange X and Y. X←→Y
XEM		812	Exchange X with memory. X←→M
MS		839	Store X in Memory. X→M
MR		1385	Recall Memory into X. M→X
LSH		168	X mantissa is left shifted while leaving decimal point in same position. Former most significant digit is saved in link digit. Least significant digit is zero.
RSH		173	X mantissa is right shifted while leaving decimal point in same position. Link digit, which is normally zero except after a left shift, is shifted into the most significant digit. Least significant digit is lost.
+	2200	6600	Add X to Y. X + Y → X. On +, −, ×, / and YX instructions, stack is popped as follows: Z→Y T→Z O→T Former X, Y are lost.
−	2200	6600	Subtract X from Y. Y − X → X
×	3200	22700	Multiply X times Y. Y × X → X
/	7800	22300	Divide X into Y. Y ÷ X → X
YX	55400	95500	Raise Y to X power. $Y^X \rightarrow X$
INV + *	1700	5000	Add X to memory. M + X → M On INV +, −, × and / instructions, X, Y, Z, and T are unchanged.
INV − *	1700	5000	Subtract X from memory. M − X → M
INV × *	2700	21400	Multiply X times memory. M × X → M
INV/*	7300	21100	Divide X into memory. M ÷ X → M
1/X	4500	22800	1 ÷ X → X. On all F (X) math instructions Y,Z,T and M are unchanged and previous X is lost.
SQRT	7000	30200	$\sqrt{X} \rightarrow X$
SQ	3000	21900	$X^2 \rightarrow X$
10X	27400	96500	$10^X \rightarrow X$
EX	30800	93900	$e^X \rightarrow X$
LN	24800	92000	ln X → X
LOG	30700	92600	log X → X
SIN	56200	95900	SIN(X) → X. On all F(X) trig functions, Y,Z,T, and M are unchanged and the previous X is lost.
COS	56200	95900	COS(X) → X
TAN	35000	97600	TAN(X) → X
INV SIN*	54000	93900	$SIN^{-1}(X) \rightarrow X$
INV COS*	54000	93900	$COS^{-1}(X) \rightarrow X$
INV TAN*	30200	92900	$TAN^{-1}(X) \rightarrow X$
DTR	9600	41700	Convert X from degrees to radians.
RTD	9600	41700	Convert X from radians to degrees.
MCLR		734	Clear all internal registers and memory; initialize I/O control signals, MDC = 8, MODE = floating point. (See initialization.)
ECLR		163	O → Error flag
JMP*		186	Unconditional branch to address specified by second instruction word. On all branch instructions, second word contains branch address to be loaded into external PC.
TJC*		208	Branch to address specified by second instruction word if JC (I_6) is true (= 1). Otherwise, skip over second word.
TERR*		191	Branch to address specified by second instruction word if error flag is true (= 1). Otherwise, skip over second word. May be used for detecting specific errors as opposed to using the automatic error recovery scheme dealt with in the section on Error Control.
TX = 0*		278	Branch to address specified by second instruction word if X = 0. Otherwise, skip over second word.
TXF*		277	Branch to address specified by second instruction word if $\|X\| < 1$. Otherwise, skip over second word. (i.e., branch if X is a fraction.)
TXLT0*		197	Branch to address specified by second instruction word if X < 0. Otherwise, skip over second word.
IBNZ		2314	M + 1 → M. If M = 0, skip second instruction word. Otherwise, branch to address specified by second instruction word.
DBNZ		2314	M − 1 → M. If M = 0, skip second instruction word. Otherwise, branch to address specified by second instruction word.
IN*		395	The processor supplies a 4-bit digit address (DA4-DA1) accompanied by a digit address strobe ($\overline{DAS}$)

decimal digits 0 through 9, the decimal point and signs for the mantissa and exponent—if floating-point format is specified.

The other two input techniques transmit data to the X register under program control. An IN instruction is executed once for entry of an entire number, while an AIN instruction is executed once per digit of a number being entered. In each case the number is entered into the X register after the stack is pushed, as illustrated for keyboard entry. Following execution for the IN or AIN instruction, digits are entered as data. Input is clocked by an output control signal accompanying the 4-bit digit address illustrated in Fig. 2.

Handshaking protocol similar to the ready-acknowledge sequence illustrated for instruction input controls data entry. Thus, it is relatively easy for any microprocessor to work asynchronously with the MM57109.

MM57109 data output is controlled by an OUT instruction which is equivalent to the IN instruction.

MM57109 data input and output philosophy contrast sharply with normal microprocessor protocol. Observe that the MM57109 requires the microprocessor to input an appropriate control command, after which the MM57109 outputs strobe signals to time data input or output. Thus, the MM57109 is not behaving like a standard peripheral device, rather, it becomes temporary bus master while inputting or outputting data.

In a normal microcomputer system, the microprocessor will input or output data from a support device just as it would for read/write memory. The device is selected via an appropriate I/O port or memory address, then a read or write control signal causes the data transfer to occur; this is how the AM9511 works.

National Semiconductor literature describes the MM57109 as either a stand-alone microprocessor or as an adjunct to

		for each digit to be input. The high order address for the number to be input would typically come from the second instruction word. The digit is input on D4-D1, using ISEL = 0 to select digit data instead of instructions. The number of digits to be input depends on the calculation mode (scientific notation or floating point) and the mantissa digit count (see Data Formats and Instruction Timing). Data to be input is stored in X and the stack is pushed (X → Y → Z → T). At the conclusion of the input, DA4-DA1 = 0.
OUT*	583	Addressing and number of digits is identical to IN instruction. Each time a new digit address is supplied, the processor places the digit to be output on DO4-DO1 and pulses the R/$\overline{W}$ line active low. At the conclusion of output, DO4-DO1 = 0 and DA4-DA1 = 0.
AIN	284	A single digit is read into the processor on D4-D1. ISEL = 0 is used by external hardware to select the digit instead of instruction. It will not read the digit until $\overline{ADR}$ = 0 (ISEL = 0 selects $\overline{ADR}$ instead of I_5), indicating data valid. F2 is pulsed active low to acknowledge data just read.
SF1	163	Set F1 high, i.e., F1 = 1.
PF1	185	F1 is pulsed active high. If F1 is already high, this results in it being set low.
SF2	163	Set F2 high, i.e., F2 = 1.
PF2	185	F2 is pulsed active high. If F2 is already high, this results in it being set low.
PRW1	130	Generates R/$\overline{W}$ active low pulse which may be used as a strobe or to clock extra instruction bits into a flip-flop or register.
PRW2	130	Identical to PRW1 instruction. Advantage may be taken of the fact that the last 2 bits of the PRW1 op code are 10 and the last 2 bits of the PRW2 op code are 01. Either of these bits can be clocked into a flip-flop using the R/$\overline{W}$ pulse.
TOGM	157	Change mode from floating point to scientific notation or vice versa, depending on present mode. The mode affects only the IN and OUT instructions. Internal calculations are always in 8-digit scientific notation.
SMDC*	163	Mantissa digit count is set to the contents of the second instruction word (= 1 to 8).
INV	166	Set inverse mode for trig or memory function instruction that will immediately follow. Inverse mode is for next instruction only.

another microprocessor. In reality, the MM57109 is not a practical stand-alone microprocessor. It should be used only in conjunction with another microprocessor because the MM57109 has no internal memory-addressing logic. A program counter, if present, must be implemented externally, using some appropriate register whose contents get triggered when appropriate timing signals are output by the MM57109. Branch instructions, though identified in Table 1, really do not exist; they simply create a control signal that external logic must use to clock an address into the external program counter.

By the time you have configured the necessary additional logic to surround a stand-alone MM57109, you will probably find it is cheaper and a good deal faster to use some simple microprocessor, even if its sole function is to monitor and control MM57109 operations.

AM9511

Now let's look at the AM9511. Functional logic for this device is illustrated in Fig. 7. The most important difference between the AM9511 and the MM57109 is that the AM9511 is a binary device. All data operations within the AM9511 handle binary data; in contrast, the MM57109 handles only BCD data. AM9511 data may be specified in fixed-point or floating-point format. Fixed-point numbers may be single- or double-precision; in each case they are treated as signed binary numbers. A single-precision fixed-point number is illustrated in Fig. 8.

This is standard signed binary data. Thus, single-precision fixed-point numbers may range in value from −32768 to +32767. Double-precision fixed-point numbers are 32 bits wide, and again use standard signed binary data format. Thus, a double-precision number may have values in the range −2147483648 through +2147483647.

Floating-point numbers are all 32 bits wide, and are interpreted as in Fig. 9. The mantissa and exponent are both binary numbers; therefore, numbers in the range $\pm(2.7 \times 10^{-20}$ to $9.2 \times 10^{-18})$ may be represented.

Observe that the AM9511 has a smaller range of valid numbers than the MM57109. You might argue that the AM9511, by handling numbers in the exponential range 10^{-20} through 10^{18}, must surely have a range adequate for any application. This is not always true.

In particular, chemical-engineering and astronomical computations frequently handle numbers outside the range allowed by the AM9511. The principal advantage of the AM9511 over the MM57109 is that the former is much faster. Table 2 summarizes AM9511 instructions. Notice that the instruction sets for the two devices are approximately

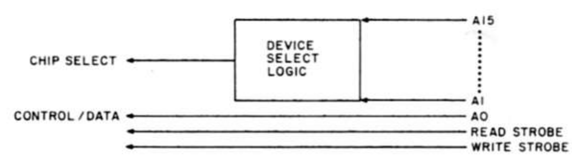


Fig. 5.

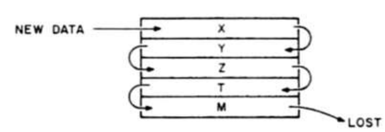


Fig. 6.

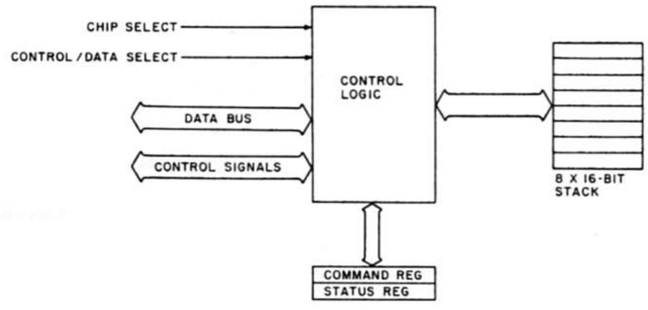


Fig. 7. AM9511 arithmetic processor functional logic.

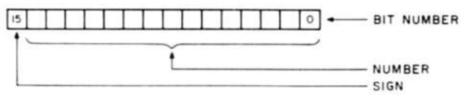


Fig. 8. A single-precision fixed-point number.

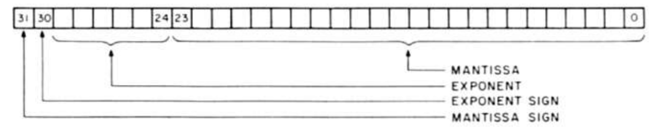


Fig. 9.

equivalent; however, based on a 500-nanosecond clock, for the AM9511 it is more than 100 times faster than for the MM57109. Also, the AM9511 is incredibly easy to incorporate into almost any microcomputer system. Control signals, data buses and address buses are typical of an 8080A support device. The AM9511 is selected via the chip-select (CS) and $C/\overline{D}$ inputs. This is the standard method used in any 8080A support device to access data control and status locations as two memory addresses or I/O ports.

The standard read and write control strobes are used to input or output data. Thus, the CS, $C/\overline{D}$, $\overline{RD}$ and $\overline{WR}$ controls together identify events as in Table 3.

Data and instructions are input via the bidirectional data bus; results and status are output via the same bus. While the AM9511 is busy executing any operation, a PAUSE signal is output low. At the end of the operation the END control signal is output low. The microprocessor acknowledges the END output by inputting EACK low.

Any command output to the AM9511 can, in addition to all other options, specify a service request to follow completion of the AM9511 operation. During a service request, CPU will process AM9511 results before initiating a new AM9511 operation. If a service request is specified, when the AM9511 completes any operation it outputs a low service-request signal. The CPU acknowledges this signal with a service-acknowledge input. Thus, the AM9511 allows the microprocessor to differentiate between an AM9511 operation that does or does not require further handling by the CPU.

CS	$C/\overline{D}$	$\overline{RD}$	$\overline{WR}$	Function
1	X	X	X	Device not selected
0	0	0	1	Read data from device
0	0	1	0	Write data to device
0	1	0	1	Read status from device
0	1	1	0	Write command to device

Table 3.

When you compare the AM9511 and MM57109 devices, selection should be based on the following trade-offs:

1. The MM57109 is a BCD device and will therefore be easier to use in a purely decimal application.

2. The MM57109 has a larger numeric range; however, you should be sure that the extensive AM9511 numeric range is insufficient before you go to the MM57109 based upon this criterion.

3. The AM9511 is significantly faster than the MM57109. There may be applications in which the AM9511 must be selected based on its speed, even if BCD-to-binary and binary-to-BCD conversions are required.

4. The AM9511 fits naturally into any 8080A microcomputer configuration; its bus and control signal interface is absolutely compatible with the 8080A. In contrast, the MM57109 is a calculator part that will need multiplexing and de-multiplexing circuits surrounding it.

Whether you choose the AM9511 or the MM57109, you will be making the right choice if your alternative is to write your own transcendental-function calculations.■

COMMAND MNEMONIC	CLOCK CYCLES	COMMAND DESCRIPTION (1)
SADD	17	Adds TOS to NOS. Result to NOS. Pop Stack
SSUB	30	Subtracts TOS from NOS. Result to NOS. Pop Stack
SMUL	92	Multiplies NOS by TOS. Result to NOS. Pop Stack
SDIV	92	Divides NOS by TOS. Result to NOS. Pop Stack
DADD	21	Adds TOS to NOS. Result to NOS. Pop Stack
DSUB	38	Subtracts TOS from NOS. Result to NOS. Pop Stack
DMUL	208	Multiplies NOS by TOS. Result to NOS. Pop Stack
DDIV	208	Divides NOS by TOS. Result to NOS. Pop Stack
FADD	56-350	Adds TOS to NOS. Result to NOS. Pop Stack
FSUB	58-352	Subtracts TOS from NOS. Result to NOS. Pop Stack
FMUL	168	Multiplies NOS by TOS. Result to NOS. Pop Stack
FDIV	171	Divides NOS by TOS. Result to NOS. Pop Stack
SQRT	800	Square Root of TOS. Result in TOS.
SIN	4464	Sine of TOS. Result in TOS.
COS	4118	Cosine of TOS. Result in TOS.
TAN	5754	Tangent of TOS. Result in TOS.
ASIN	7668	Inverse Sine of TOS. Result in TOS.
ACOS	7734	Inverse Cosine of TOS. Result in TOS.
ATAN	6006	Inverse Tangent of TOS. Result in TOS.
LOG	4490	Common Logarithm (base 10) or TOS. Result in TOS.
LN	4478	Natural Logarithm (base e) of TOS. Result in TOS.
EXP	4616	Exponential (e^x) of TOS. Result in TOS.
PWR	9292	NOS raised to the power in TOS. Result to NOS. Pop Stack.
NOP	4	No Operation
FIXS	92-216	Converts TOS from floating-point to single-precision fixed-point format.
FIXD	100-346	Converts TOS from floating-point to double-precision fixed-point format.
FLTS	98-186	Converts TOS from single-precision fixed-point to floating-point format.
FLTD	98-378	Converts TOS from double-precision fixed-point to floating-point format.
CHSS	26	Changes sign of single-precision fixed-point operand on TOS.
CHSD	34	Changes sign of double-precision fixed-point operand on TOS.
CHSF	16	Changes sign of floating-point operand on TOS.
PTOS	16	Push single-precision fixed-point operand on TOS to NOS.
PTOD	20	Push double-precision fixed-point operand on TOS to NOS.
PTOF	20	Push floating-point operand on TOS to NOS.
POPS	10	Pop single-precision fixed-point operand from TOS. NOS becomes TOS.
POPD	12	Pop double-precision fixed-point operand from TOS. NOS becomes TOS.
POPF	12	Pop floating-point operand from TOS. NOS becomes TOS.
XCHS	18	Exchange single-precision fixed-point operands TOS and NOS.
XCHD	26	Exchange double-precision fixed-point operands TOS and NOS.
XCHF	26	Exchange floating-point operands TOS and NOS.
PUPI	16	Push floating-point constant "π" onto TOS. Previous TOS becomes NOS.

Notes: 1. *Nomenclature:* TOS is Top Of Stack. NOS is Next On Stack.
2. All derived floating-point functions destroy the contents of the stack. Only the result can be counted on the be valid upon command completion.
3. Format conversion commands (FIXS, FIXD, FLTS, FLTD) require that floating-point data format be specified (command bits 5 and 6 must be 0).

Table 2. AM9511 instruction description table.

Ken Barbier
Borrego Engineering
PO Box 1253
Borrego Springs CA 92004

Money Manipulations

keep ahead of those cash-flow problems

```
1 REM    CASH FLOW PROGRAM         29 JULY 77
7 A$="$$#######. ##"
8 B$="#####"
9 PRINT:PRINT:PRINT
10 PRINT"INVESTMENT MINUS DRAW"
11 PRINT"                    QUARTERLY STATEMENT"
15 PRINT
20 PRINT"PRINCIPAL: $; INTEREST: %/YR; DRAW: $/MO."
21 PRINT" INTEREST EARNED SHOWN BY QUARTER TOTAL"
22 PRINT" DRAW IS CURRENT MONTHLY RATE, INFLATED"
25 PRINT:PRINT
30 M=0
31 Y=0
35 C=2
40 INPUT "PRINCIPAL=";P
50 INPUT "INTEREST=";I
60 INPUT "DRAW=";D
61 INPUT "INFLATION=";A
62 A=A/1200
70 I=I/1200
80 PRINT:PRINT:PRINT
90 PRINT"MONTH  PRINCIPAL  EARNED  DRAW"
100 E=P*I
110 P=P+E
120 P=P-D
125 IF P<0 THEN 190
130 M=M+1
135 D=D*(1+A)
137 Y=Y+E
138 IF M=123 THEN 180
140 IF M>240 THEN 1200
141 IF C=0 GOTO 144
142 C=C-1
143 GOTO 100
144 C=2
146 PRINT USING B$;M;:PRINT" ";
150 PRINT USING A$;P,Y,D
151 Y=0
170 GOTO 100
180 PRINT:PRINT:PRINT: PRINT
181 GOTO 141
190 PRINT:PRINT:PRINT:PRINT:PRINT
1200 END
```

Fig. 1. Cash Flow program listing. Written in 12K Extended BASIC, it can be run on smaller BASICs by changing lines 146 and 150 to: 150 PRINT M,P,Y,D. Lines 7 and 8 can then be dropped. The output will not be so nicely formatted, however.

You say you're getting ready to punch your boss's lights out, but you're not sure your life savings will support the wife and kiddies until you get out of jail? Or maybe you're just getting old (like me) and think it's time to retire, but you want to be sure you have enough loot stashed away to supplement Uncle Sam's pittance and provide enough to live on – forever and ever. Or perhaps you are ready to throw in the towel at the boiler factory and open your own computer store . . . and want to know how long you can hold out until the first cash customer comes walking in. Well, tell you what I'm going to do . . .

Computing Cash Flow

The Cash Flow program listing in Fig. 1 assumes that an initial investment is made at a fixed rate of interest (compounded monthly). But instead of simply figuring compound interest, Cash Flow assumes that we will be drawing on these reserves, for reasons such as those listed above. Furthermore, life being what it is, the amount we have to withdraw will be subject to inflation, so the program takes this factor into account as well. Since Uncle Sam insists we pay income tax on the interest paid on our investments, we will also need a statement showing interest earned. While the program will not fill out your income tax form for you, it will, considering all these factors, tell you how long your loot will last.

For example, let's take a look at a typical Cash Flow run (Fig. 2). Dick and Jane have both been working and diligently squirreling money away. They have accumulated forty kilobucks and would like to use it to finance an early retirement. What they need to know is whether or not the money will hold out until social security helps them out (assuming it doesn't go broke first).

Being conservative, they will invest the money in insured savings, which, for our example, we will assume pays 5.75 percent per year, compounded monthly. They have moved into a less expensive house, but there are still payments to make. Now, our couple must figure the maximum amount per month that they will have to draw from their savings to live on. This fictional account shows that they have arrived at a figure of $750 per month, which certainly should be enough to feed two mouths.

Next, we throw in a little magic. D. and J. have consulted their financial expert, and he assures them they can expect an inflation rate of 3 percent per annum to apply to the commodities they will be consuming. This figure sounds low today, but if coffee, new cars, etc., are avoided it is not too unrealistic.

All the above conditions established, we load Cash Flow, which is written in Altair BASIC, 12K Extended, Version 3.2. Instructing it to run, we are informed that we will be provided with a quarterly statement, and we are asked to enter the amount of principal (in dollars); the interest rate (in percent per annum); the amount we wish to withdraw (in dollars per month); and the expected annual rate of inflation. Having received these variables, Cash Flow proceeds to produce the quarterly-statement table shown in Fig. 2.

Since this is a quarterly statement, the number of the month for which the figures apply will increment by three. The amount of principal remaining at the end of that month is shown in the next column. The third column shows the total interest earned for the previous quarter, which is what we will have to pay income tax on. This last column shows our draw for the current month. This amount always increases because we have to assume that inflation will continue to spiral.

When all of the money is used up, Cash Flow will terminate, and we will have to go back to work. We see that Dick and Jane can survive for about five years. Well, maybe they'd better try to cut costs a little. Then we can try the program again, using a lower Draw figure.

When this program was first run, the nice round numbers in the cents column under Principal raised suspicion. The BASIC manual states that single-precision numbers are printed with a maximum of six decimal digits, and we are asking BASIC to work with seven digits! So, we should add the following line to our program: 2 DEFDBL P.

Now when we run the program with the same variables, we get the output shown in Fig. 3, since Principal is computed in double precision. We can see the pennies and nickles, but the results don't change! This is because we had sufficient accuracy to begin with, the internal representation of our principal being in binary bits, which don't exactly relate evenly to six-decimal digits. Our initial accuracy was *barely* sufficient, though, so it would be a good idea to leave the second line in our program, in case a rich uncle dies and leaves more money to play with.

Since Dick and Jane are only 23 years old (surprise!) they have decided to postpone the early retirement and keep on working and saving. Now they can use the same program to estimate how their savings will grow if left untouched. If no money is drawn from the investment, Cash Flow becomes a straightforward compound-interest program, as we can see in Fig. 4.

Here, we set draw and inflation to zero, and Cash Flow gives a quarterly statement of earnings and accumulation for our savings account. The program gets tired and quits after 20 years. Dick and Jane probably will, too! ■

```
INVESTMENT MINUS DRAW
          QUARTERLY STATEMENT

PRINCIPAL:   $;   INTEREST: %/YR;     DRAW:  $/MO.
   INTEREST EARNED SHOWN BY QUARTER TOTAL
   DRAW IS CURRENT MONTHLY RATE, INFLATED

PRINCIPAL=? 40000
INTEREST=? 5.75
DRAW=? 750
INFLATION=? 3

MONTH        PRINCIPAL       EARNED       DRAW
  3          $38311.30       $566.95      $755.64
  6          $36581.20       $542.48      $761.32
  9          $34808.90       $517.41      $767.05
 12          $32993.80       $491.73      $772.81
 15          $31135.00       $465.43      $778.62
 18          $29231.70       $438.49      $784.48
 21          $27283.30       $410.92      $790.38
 24          $25289.00       $382.69      $796.32
 27          $23247.80       $353.80      $802.31
 30          $21159.10       $324.23      $808.34
 33          $19022.00       $293.97      $814.42
 36          $16835.70       $263.02      $820.54
 39          $14599.20       $231.35      $826.71
 42          $12311.90       $198.96      $832.93
 45           $9972.66       $165.83      $839.19
 48           $7580.74       $131.95      $845.50
 51           $5135.21        $97.31      $851.86
 54           $2635.15        $61.90      $858.26
 57             $79.62        $25.69      $864.71
```

Fig. 2. Sample Cash Flow run. This printout shows how long an initial investment of $40,000 will last while earning 5.75 percent interest, but being drawn on at the rate of $750 per month, inflated 3 percent per year.

```
INVESTMENT MINUS DRAW
          QUARTERLY STATEMENT

PRINCIPAL:   $;   INTEREST: %/YR;     DRAW:  $/MO.
   INTEREST EARNED SHOWN BY QUARTER TOTAL
   DRAW IS CURRENT MONTHLY RATE, INFLATED

PRINCIPAL=?  40000
INTEREST=?   5.75
DRAW=? 750
INFLATION=? 3

MONTH        PRINCIPAL       EARNED       DRAW
  3          $38311.32       $566.95      $755.64
  6          $36581.21       $542.48      $761.32
  9          $34808.94       $517.41      $767.05
 12          $32993.78       $491.73      $772.81
 15          $31134.96       $465.43      $778.62
 18          $29231.74       $438.49      $784.48
 21          $27283.34       $410.92      $790.38
 24          $25288.97       $382.69      $796.32
 27          $23247.83       $353.80      $802.31
 30          $21159.12       $324.23      $808.34
 33          $19022.01       $293.97      $814.42
 36          $16835.66       $263.02      $820.54
 39          $14599.24       $231.35      $826.71
 42          $12311.86       $198.96      $832.93
 45           $9972.66       $165.83      $839.19
 48           $7580.75       $131.95      $845.50
 51           $5135.22        $97.31      $851.86
 54           $2635.15        $61.90      $858.26
 57             $79.62        $25.69      $864.71
```

Fig. 3. A double-precision run. The net results have not changed, but would for larger principals. Double precision results in a more accurate printout, but the program takes longer to run.

```
INVESTMENT MINUS DRAW
          QUARTERLY STATEMENT

PRINCIPAL:   $;   INTEREST:   %/YR;   DRAW:  $/MO.
   INTEREST EARNED SHOWN BY QUARTER TOTAL
   DRAW IS CURRENT MONTHLY RATE, INFLATED

PRINCIPAL=? 10000
INTEREST=? 6
```

```
DRAW=? 0
INFLATION=? 0
```

MONTH	PRINCIPAL	EARNED	DRAW
3	$10150.75	$150.75	$0.00
6	$10303.78	$153.02	$0.00
9	$10459.11	$155.33	$0.00
12	$10616.78	$157.67	$0.00
15	$10776.83	$160.05	$0.00
18	$10939.29	$162.46	$0.00
21	$11104.20	$164.91	$0.00
24	$11271.60	$167.40	$0.00
27	$11441.52	$169.92	$0.00
30	$11614.00	$172.48	$0.00
33	$11789.08	$175.08	$0.00
36	$11966.81	$177.72	$0.00
39	$12147.21	$180.40	$0.00
42	$12330.33	$183.12	$0.00
45	$12516.21	$185.88	$0.00
48	$12704.89	$188.68	$0.00
51	$12896.42	$191.53	$0.00
54	$13090.83	$194.42	$0.00
57	$13288.18	$197.35	$0.00
60	$13488.50	$200.32	$0.00
63	$13691.84	$203.34	$0.00
66	$13898.25	$206.41	$0.00
69	$14107.77	$209.52	$0.00
72	$14320.44	$212.68	$0.00
75	$14536.33	$215.88	$0.00
78	$14755.46	$219.14	$0.00
81	$14977.90	$222.44	$0.00
84	$15203.70	$225.79	$0.00
87	$15432.89	$229.20	$0.00
90	$15665.55	$232.65	$0.00
93	$15901.71	$236.16	$0.00
96	$16141.43	$239.72	$0.00
99	$16384.76	$243.33	$0.00
102	$16631.76	$247.00	$0.00
105	$16882.49	$250.73	$0.00
108	$17136.99	$254.51	$0.00
111	$17395.34	$258.34	$0.00
114	$17657.57	$262.24	$0.00
117	$17923.76	$266.19	$0.00
120	$18193.97	$270.20	$0.00
123	$18468.24	$274.28	$0.00
126	$18746.65	$278.41	$0.00
129	$19029.26	$282.61	$0.00
132	$19316.13	$286.87	$0.00
135	$19607.32	$291.19	$0.00
138	$19902.91	$295.58	$0.00
141	$20202.95	$300.04	$0.00
144	$20507.51	$304.56	$0.00
147	$20816.66	$309.15	$0.00
150	$21130.47	$313.81	$0.00
153	$21449.02	$318.54	$0.00
156	$21772.37	$323.35	$0.00
159	$22100.59	$328.22	$0.00
162	$22433.76	$333.17	$0.00
165	$22771.95	$338.19	$0.00
168	$23115.24	$343.29	$0.00
171	$23463.70	$348.47	$0.00
174	$23817.42	$353.72	$0.00
177	$24176.47	$359.05	$0.00
180	$24540.94	$364.46	$0.00
183	$24910.89	$369.96	$0.00
186	$25286.43	$375.53	$0.00
189	$25667.62	$381.20	$0.00
192	$26054.57	$386.94	$0.00
195	$26447.34	$392.78	$0.00
198	$26846.04	$398.70	$0.00
201	$27250.75	$404.71	$0.00
204	$27661.55	$410.81	$0.00
207	$28078.56	$417.00	$0.00
210	$28501.84	$423.29	$0.00
213	$28931.51	$429.67	$0.00
216	$29367.66	$436.15	$0.00
219	$29810.38	$442.72	$0.00
222	$30259.78	$449.40	$0.00
225	$30715.95	$456.17	$0.00
228	$31178.99	$463.05	$0.00
231	$31649.02	$470.03	$0.00
234	$32126.13	$477.11	$0.00
237	$32610.44	$484.31	$0.00
240	$33102.04	$491.61	$0.00

Fig. 4. Compound-interest run. If Draw is set to zero, Cash Flow becomes a straight compound-interest computation. Here, $10,000 was invested at 6 percent for 20 years. Changing program line 140 can vary this time limit.

THE MICRO WORKS

COME ON UP AND SEE ME SOMETIME

Give your 6800 computer the gift of sight! The Micro Works Digisector® opens up a whole new world for your computer. Your micro can now be a part of the action, taking pictures like this one to amuse your friends, watching your home while you're away, helping your household robot avoid bumping into walls, providing fast to slow scan conversion for you hams . . . the applications abound.

The Micro Works Digisector is a completely unique device; its resolution and speed are unmatched in industry and the price is unbeatable anywhere. The Digisector and a cheap TV camera are all you'll need to see eye to eye with your 6800. Since operation is straightforward, you don't have to be a software wizard to utilize the Digisector's extensive capabilities. The Micro Works Digisector board provides the following exclusive features:

- High Resolution—a 256 x 256 picture element scan
- Precision—64 levels of grey scale
- Speed—Conversion times as low as 3 microseconds per pixel
- Versatility—Accepts either interlaced (NTSC) or non-interlaced (Industrial) video input
- Compactness—Utilizes 1 I/O slot in your SWTPC 6800 or equivalent
- Economy—The Digisector is a professional tool priced for the hobbyist

The Digisector (DS-68), like all Micro Works products, comes fully assembled, tested and burned in. Only the highest quality components are used, and the boards are double sided with plated through holes, solder mask and silkscreen. All software is fully source listed and commented. The Micro Works is proud to add the DS-68 to its line of quality computer accessories for the hobbyist. **Price 169.95**

The Micro Works 6800 series of computer accessories also includes:

PSB-08 PROM System Board	119.95
regulated + 12 volts	124.95
B-08 2708 EPROM Programmer	99.95
regulated + 12 volts	104.95
U2708 EPROM Software	29.95
Casette tape	9.95
UIO Universal I/O Board	24.95
X-50 Extender Board	29.95
X-30 Extender Board	22.95

Visa and Master Charge Accepted

P.O. BOX 1110 DEL MAR, CA. 92014 714-756-2687

M31

A Name To Remember...

Wasatch

SEMICONDUCTOR PRODUCTS

for

44 PIN 5 VOLT
4.5" x 6.5" CARDS

RAM, PROM, EPROM, PARALLEL I/O, SERIAL I/O, A/D, D/A, CPU

Watch For New Product Releases

W13

25 SOUTH 300 EAST · SUITE 215 · SALT LAKE CITY, UTAH 84111 · 714/752-1374

Strings and Things

BASIC conversion techniques

Richard Roth
TSA Software
5 N. Salem Road
Ridgefield CT 06877

You have advanced far enough in programming to use character strings; yet, when you try to run a program using character strings from a book or article, you find half of them don't make any sense. If so, or if you are interested in handling characters in general, this article is for you.

A character string, basically a one-dimensional array or vector of characters, is a sequence of characters one after another. What distinguishes it from a vector of numbers is that it is used as a whole, rather than a character at a time.

In a game, the program may ask for someone's name, but it doesn't care that JOHN is a J followed by an O, H, N. The individual letters are considered a unit. In contrast, a mailing-list program that prints a list by last names scans MARYƀƀJ.ƀ-JONES to find the last word. It does this by scanning the characters until it finds a sequence of characters followed only by blanks. A space (represented by ƀ or blank) breaks the sequence of characters that comprise a word. We call such a break character a *delimiter.* Commas and periods also break the sequences of words into smaller units—phrases and sentences. A smaller unit of a character string is called a *substring*. Another special feature of character strings is length; a unit called NAME can vary from ED to STASTICOVICH. Usually, we fix a maximum length, but often we want to know the current length.

```
DIM AA(3)
READ AA(1), AA(2), AA(3)
DATA 'SALLY', 'JOE', 'SPOT'
PRINT 'A GIRL IS', AA(1)
PRINT 'A BOY IS', AA(2)
PRINT 'A DOG IS', AA(3)
```

Example 1.

HP BASIC	**DEC BASIC**
HP	DEC
Data General (DG)	Mits/Microsoft
North Star	BASIC-E
Computer Science Corp.	Tymshare
	Micro-polius (??)

Table 1.

The problem arises when you want to use strings in BASIC, originally intended to work with numbers. Of course, a letter can be represented by a number, such as A = 1, B = 2 . . . or by the ASCII character set. In ASCII, digits (0-9), letters and special characters (such as Bell or Return) are all represented by a single integer from 0 through 127 (funny—it just fits in one byte!). In working with such simple numbers, BASIC wastes space because it is prepared for many digits of precision and doesn't know how simple a number is. Dealing with varying length and the string as a unit requires some built-in features. In the scientific language FORTRAN, the programmer must have a whole set of special subroutines to deal with strings.

When BASIC was first developed by Dartmouth's Kemeny and Kurtz, the only strings allowed were literals in print statements for title and labels such as: 100 PRINT "X = ",X. In early versions of BASIC, such as GE-635 Mark I Timesharing, extensions were added to allow the storage of strings, which were handled like single numbers. However, no advanced capability was available. A string array was specified by giving it a two-letter name. All one could do was print the

```
HP                                       DEC
100 DIM N$(30),L$(10)                    100 REM
110 S≠0 (state is beginning)             110 S=0
120 C=1 (character 1)                    120 C=1
130 N$="SALLYƀƀJ.ƀJONES"                 130 N$="SALLYƀƀJ.ƀJONES"
140 IF C>LEN(N$) GO TO 200               140 IF C>LEN(N$) GO TO 200
150 IF N$(C,C)="ƀ"THEN S=C               150 IF MID$(N$,C,1)="ƀ"THEN  S=C
160 C=C+1                                160 C=C+1
170 GO TO 140                            170 GO TO 140
200 REM Now S=Char of last space         200 REM
210 L$=N$(S+1)                           210 L$=RIGHT$(L$,S+1)
```

Example 2.

whole string. (See Example 1.)

This extension was short lived, but it set the stage for what we now have. The idea of uniquely specifying a string name became more prevalent, and '$' was finally accepted as the last character of a string name. But we are still plagued by the questions: How does one specify a single character of a string; and how does one specify a matrix of strings? Two primary approaches developed—one by Digital Equipment Corporation (DEC) and the other by Hewlett-Packard (HP) in the HP-200 series.

DEC emphasized many strings grouped as a matrix; and so A$(1) became the first element in a string matrix. HP emphasized each character in the string; and so A$(1) became the first character of string A$. These approaches led to a major difference in string handling. DEC BASIC requires a special way of getting a single character of a string, while HP BASIC must handle a string array specially. Table 1 shows a summary of the different BASICs.

From now on I will refer to the two schemes simply as HP or DEC, even though most schemes I will be referring to have not been written by either company.

What's It All Mean?

The issue involves how one deals with strings. For simple strings, such as printing the name of a single game-player, there is (almost) no difference (see Table 2).

Since HP BASIC uses the subscript notations to refer to substrings, DIM specifies the length of the string. DEC BASIC uses DIM to indicate how many strings in a string matrix; if no DIM occurs, it is just a single string (also called a scalar string, as opposed to a matrix).

DEC BASIC has no way of specifying maximum string length. They allow a maximum limit, usually 255 characters, set by the BASIC designer. HP BASIC tends to allow length limited only by memory size. Since HP BASIC knows how big a string can get, it can reserve a fixed space. DEC BASIC must constantly shift the strings around as lengths change. In this respect, HP BASIC enjoys a speed advantage.

Character Manipulations

There are two levels of manipulations—character and string. Each scheme of BASIC has its own home ground: character for HP and string for DEC. Getting at a single character is required for many functions. An early example suggested extracting a last name to alphabetize a mailing list. Example 2 shows this in both schemes.

To get at the fifth character of a string, HP BASIC uses only N$(5,5), while DEC BASIC uses MID$(N$,5,1). In our example, the program considers one character at a time from the string N$ with the name, until it gets to the end. Each time it sees a blank, it saves the character number in S. With no trailing blanks, the program, when it reaches line 200, S will point to the last blank. So the last name is S+1 through the end. (For simplicity, I am assuming a statement can follow an IF-THEN statement, as in most current BASICs.)

Table 3 shows how to get at a specific character. By HP rules the first subscript is the starting character, the second is ending character: if A$ = "ABCD", then A$ (2,3) = "BC". If no second subscript exists, then the rest of the string is used: if A$ = "ABCDEF" then A$(3) = "CDEF". DEC uses functions MID$, RIGHT$, LEFT$ (as shown in the table) for the string A$ = "ABCDEF".

Single-character string functions are shown in Table 4.

String Manipulations

While DEC BASIC has awkward functions when dealing with substrings, HP BASIC has a far greater problem when many strings must be manipulated. It has no way to handle a group of strings of variable length. In the HP-3000 BASIC this was remedied in an elegant manner—especially true to BASIC syntax; unfortunately, only HP-3000 and Computer Science have implemented this

```
HP                          DEC
DIM A$(30)
INPUT A$                    INPUT A$
PRINT "HELLO", A$           PRINT "HELLO", A$

RUN                         RUN
?SAM (CR)                   ?SAM (CR)
HELLO  SAM                  HELLO  SAM
```

Table 2.

DEC	HP
MID$(A$, starting char, length)	
MID$(A$,2,2)="BC"	A$(2,3)
LEFT$(A$, length)	
LEFT$(A$,3)="ABC"	A$(1,3)
RIGHT$(A$, starting char)	
RIGHT$(A$,3)="CDEF"	A$(3)

(Note: The parameter for RIGHT$ is starting character in DEC BASIC-plus, but length from the right in Mits BASIC and BASIC-E.)

Table 3.

Function		HP	DEC
Length		LEN(A$)	LEN(A$)
Substring	- 1 char at I	A$(I,I)	MID$(A$,I,1)
	- N chars at I	A$(I,I+N−1)	MID$(A$,I,N)
	- Char I to J	A$(I,J)	MID$(A$,I,J−I+1)
	- Char I to end	A$(I)	RIGHT$(A$,I)
	- Char 1 to Char I	A$(1,I)	LEFT$(A$,I)

Table 4.

in their BASICs. They have added:

```
DIM S(3)$(5)
A$=S(1)$(3,4)
```

The first subscript is a matrix; the second, a substring. Similarly, the first DIM value is matrix size; the second is maximum length.

Most people with HP BASIC can handle (with difficulty) a form of string matrix. Imagine the string V$, length 100, to be made of ten substrings, each ten characters long. The key is to fill out each string to a full ten characters; otherwise, the larger string will have holes. Creating one of those holes by putting in a shorter string will chop off the rest of the larger string. This also makes all the pseudomatrix elements a fixed length, which is annoying but better than no string matrix. For example, the fifth string in string matrix V$ is extracted by using: S5$ = V$(4*L + 1,5*L) (this is for a matrix starting at element 1). Two simple user functions will ease this calculation (see Example 3).

Concatenation

The second major function in string manipulation is concatenation, i.e., combining two strings to make one. For example, "HEL" + "LO" = "HELLO" (using DEC concatenation operator). HP has no common, direct way of doing this. Both + and , are allowed in some HP BASICs as concatenation operators. If no operator exists, HP BASIC allows a rather strange use of the subscript/substring to do this (see Table 5). At the L1 and L2 calculations, X$ is kept at full length and need not be refilled.

When using HP form strings for pseudostring matrices or concatenation, one must be very careful to fill out each string assignment where the subscript/substring is on the left side, i.e., S$(1,4) = A$. An improper assignment may chop off the end of the string on the left. This varies between HP-style BASICs, for example:

```
A$(5,9)=B$   where LEN(B$)<4
A$(5)=B$
```

In both cases, the length of A$ might become 5 + LEN(B$). (Data General had this problem before Release 3 RDOS BASIC, whereas, North Star Release 2 does not have the problem.)

Commands, Special Characters, Numbers and Input/Data

There are several less important differences that relate to assorted areas that vary between both schemes, all versions. Commands vary from BASIC to BASIC, for example, NEW or SCR (scratch), which is used to clear out an old program.

Getting special characters into and out of strings requires special care. Normally, a bell, for example, cannot be entered into a string. Some BASICs allow the code to be typed in a quoted literal. This can cause a problem because a listing will not show the character or, even worse, it will do the function (for example, turn on the paper-tape punch). One scheme by DG allows a special form in literal <#> in which the number is the internal form of the special code. For example, <7> is an ASCII BEL Code. The more common version allows a function, usually CHR$, that converts the numeric value to a string of the same character (BELL Code = CHR$(7)). The reverse function is ASC for ASCII value, where ASC("A") = 65 (the value of the letter A in the ASCII code). (Some BASICs use an ASCII null (true 0 byte) to indicate the end of a string. So A$(10) = CHR$(0) will chop off the string at 9 characters—if your BASIC does this.)

A similar conversion from internal to character string form is often available for numbers, too. NUM$(A) = "0.0" if A = 0 or VAL(A$) = 0 if A$ = "0.0". If your BASIC does not have a formatted print, these are useful in doing special output or input formatting. Read *your* manual before trying these functions; they might not do what you would expect. Depending on the BASIC, the following sequence could give a lot of trouble.

```
10 A=10
20 A$=NUM$(A)
30 F$="FILE"+A$
40 OPEN FILE F$
```

Some BASICs format a "NUM$" call exactly like output and put a space before the numeric string. For example, F$ = "FILEƀ10"—not "FILE10" Some special functions allow any string, expression or literal, while others must be a simple variable. (The difference between internal form of a number and ASCII byte or a character string can be confusing for the novice. 10 is not the same as "10" and if you are not sure why, find someone who knows. For example, a BEL code is an ASCII 7, not 7.0 or "7"—the difference depends on the function required.) Because it is not clear which is the "obvious way," both exist (see Example 4). DEC style says when the IF condition is true execute the rest of the statement; if it is false, continue on the next line.

```
DEF FNL(X)=(X-1)*L+1/DEF FNH(X)=(X*L)
A$(FNL(X),FNH(X)) references element X where:
X=subscript, L=length and A$ is pseudomatrix.
```

Example 3.

```
     HP
100  IF A=B THEN PRINT "EQUALS"/GO TO 300
110  GO TO 400

     DEC
100  IF A=B THEN PRINT "EQUALS": GO TO 300
110  GO TO 400
```

Example 4.

```
            (from DEC BASIC-PLUS
A$="BCDEFAF"
INSTR(1,A$,"AF")=6       (6th char position)
INSTR(1,A$,"ABD")=0      (not found)
INSTR(6,A$,"F")=7        (start looking at 6th char)
```

Example 5.

```
    DEC                           HP
10  A$="HEL"                  10  DIM X$(80)
20  B$="LO"                   20  X$="                    " (80 blanks)
30  S$=A$+B$                  30  A$="HEL"
40  PRINT "STRING=",S$        40  B$="LO"
    Run                       50  L1=LEN(A$)
    STRING=HELLO              60  L2=LEN(B$)
                              70  X$(1,L1)=A$
                              80  X$(L1+1,L1+L2)=B$
                              90  S$=X$(1,L1+L2)
                              100 PRINT "STRING=",S$
                                  Run
                                  STRING=HELLO
```

Table 5.

```
5-190   — Dimension and Functions
200-299 — Read in names and Data Statements
300-499 — Swap names, last name first
500-699 — Bubble sort alphabetically
700-899 — Print sorted list
```

Table 6.

Most HP BASICs only allow a line number after THEN. North Star says if true, execute the rest; if false, skip only the THEN clause, not the line. HP-style BASIC may or may not print "EQUALS," but it will always go to line 300; DEC style will only go to line 300 if "EQUALS" is printed; otherwise it will go to line 400.

Another feature of some BASICs is a string search, which locates a substring in a larger string (see Example 5).

Back to Reality

Let's condense all this discussion into one example which compares a list sort in HP-style and DEC-style BASIC. To add character functions we enter the list first name first and sort it first name last. Both are listed in Programs A and B and have approximately corresponding line numbers (see Table 6).

For the HP-like BASIC, we used North Star BASIC, which took 22 seconds from run to ready; the DEC-like BASIC was BASIC-E, which took 10 seconds (but it's a partial compiler). Neither time reflects a great sort but it works and illustrates our discussion here. (Fig. 1 is a run of the program.)

Peculiarities of the HP-like version are primarily related to the pseudomatrix required because the names functions FNL and FNH are used to calculate the start and end charac-

```
READY
LIST
100  REM WRITTEN IN NORTHSTAR BASIC (RELEASE 2)
110  READ N9
120  DIM N$(N9*30), F$(30),F1$(30),F2$(30),A$(30)
130  REM USE FUNCTIONS FOR PSEUDOMATRIX OF STRINGS
140  DEF FNL(X)=(X-1)*30+1 \ DEF FNH(X)=X*30
150  DEF FNA$(A$)
160  IF LEN(A$)>=30 THEN RETURN A$
170  A$=A$+"ƀ" \ GOTO 160 \ FNEND
200  REM IN NAMES
205  PRINT " **** NAMES ****" \ PRINT
210  N$="" \ REM CLEAR MATRIX
220  FOR I=1 TO N9
230  READ F$
235  PRINT F$
240  F$=F$+"$" \ REM MARK END OF NAME FOR REVERSE ROUTINE
250  F$=FNA$(F$)  \ REM FILL NAME TO 30 CHARS
260  N$=N$+F$
270  NEXT I
280  REM DATA
282  DATA 10
284  DATA "SALLY JONES", "SAM SMITH", "JOE SMITH", "TIM CAMBELL", "ED HILL"
286  DATA "STEVE MOODY", "ROGER HEAD", "SHIRLEY JONES", "ISSAC DEAR", "RICH KING"
300  REM RE-ORDER LAST NAME FIRST
310  FOR N1=1 TO N9
320  F$=N$(FNL(N1),FNH(N1))
330  C=1
335  REM LOOP UNTIL END MARK FOUND
340  IF F$(C,C)="$" THEN 380
350  IF F$(C,C)="ƀ" THEN S=C
360  C=C+1
370  GOTO 340
380  REM REVERSE FIRST & LAST NAMES
390  F1$=F$(1,S-1)          \ REM FIRST NAME
400  F2$=F$(S+1,C-1)        \ REM LAST NAME
410  F$=F2$+","+F1$
415  REM PUT BACK IN MATRIX (NOTE FULL 30CHARS SO NO LEFT-OVERS)
420  N$(FNL(N1), FNH(N1))=FNA$(F$)
430  NEXT N1
500  REM BUBBLE SORT, LOOP UNTIL NO SWAP ON A PASS
510  F=0
520  FOR I=2 TO N9
530  IF N$(FNL(I),FNH(I))>=N$(FNL(I-1),FNH(I-1)) THEN 590
540  REM SWAP
550  F=1      / REM REMEMBER A SWAP WAS DONE
560  F$=N$(FNL(I),FNH(I))
570  N$(FNL(I),FNH(I))=N$(FNL(I-1),FNH(I-1))
580  N$(FNL(I-1),FNH(I-1))=F$
590  NEXT I
600  IF F>0 THEN 510      \ REM KEEP TRYING TILL NO SWAPS
800  REM PRINT SORTED LIST
805  PRINT \ PRINT \ PRINT " **** SORTED NAMES ****" \ PRINT \ PRINT
810  FOR I=1 TO N9
820  F$=N$(FNL(I),FNH(I))
830  PRINT F$
840  NEXT I
850  END
READY
```

Program A. Mailing list (HP style).

```
MAILING. BAS WRITTEN IN BASIC-E (11/6/77)
5    REM WRITTEN IN BASIC-E
7    REM GET NUMBER OF NAMES
10   READ N9
15   DIM N$(N9)
200  REM READ IN NAMES
205  PRINT " **** NAMES ****"
210  FOR I=1 TO N9
220  READ N$(I)
225  PRINT N$(I)
230  NEXT I
240  DATA 10
250  DATA SALLY JONES, SAM SMITH, JOE SMITH, TIM CAMBELL, ED HILL
260  DATA STEVE MOODY, ROGER HEAD, SHIRLEY JONES, ISSAC DEAR, RICH KING
300  REM RE-ORDER LAST NAME FIRST
310  FOR N1=1 TO N9
320      C=1 : F$=N$(N1) : L=LEN(F$)
325      REM LOOP UNTIL LAST CHAR AND MARK LAST BLANK
330      IF C>L THEN 365
340          IF MID$(F$,C1)="ƀ" THEN S=C
350          C=C+1
360          GOTO 330
365      REM ACTUALLY SHUFFLE NAMES
370      F1$=LEFT$(F$,S-1)        : REM FIRST NAME
379          REM NOTE RIGHT$(NAME,LENGTH)
380      F2$=RIGHT$(F$,L-S)       : REM LAST NAME
390      F$=F2$+","+F1$
392          REM FILL OUT LENGTH SINCE 3 CHAR STR<4 CHAR STR
395      N$(N1)=F$+LEFT$("                              ",30-LEN(F$))
400  NEXT N1
500  REM DO SIMPLE BUBBLE SORT
510  F=0     : REM LOOP UNTIL NO SWAPS ON A PASS
520  FOR I=2 TO N9
530      IF N$(I)>N$(I-1) THEN 590
540          REM SWAP
550          F=1      : REM REMEMBER SWAP
560          F$=N$(I)
570          N$(I)=N$(I-1)
580          N$(I-1)=F$
590  NEXT I
600  IF F>0 THEN GOTO 500       : REM TEST FOR DONE
800  REM PRINT SORTED LIST
810  PRINT : PRINT : PRINT " **** SORTED LIST ****"
820  FOR I=1 TO N9
830  PRINT N$(I)
840  NEXT I
```

Program B. Mailing list (DEC style).

ters of a name element of 30 characters in the pseudomatrix N$ of names. FNA$ is used to fill a name out to 30 characters. Since a pseudomatrix element must be a fixed length, $ is used at the end of a name on initial entry so the first name/last name swap tells where the name ends.

The DEC-style version looks much nicer, primarily because it accepts a tab character while being typed in and thus is easier to format (called pretty-print). It is wise to do this if you can since it makes the reading of the program easier.

Line 380 uses the RIGHT$ function. This particular BASIC has the second parameter as the length, so RIGHT$ returns the right n-most characters (i.e., RIGHT$("ABCDEF",3) = "DEF"). Yet a true DEC-written BASIC will return from the nth character to the end (i.e., RIGHT$("ABCDEF",3) = "CDEF"). Line 395 illustrates one of the nice things about a DEC-like BASIC—string elements of variable length. This particular BASIC says a long string of As is greater than a short string of Bs, i.e., AAA> BB. Well, to each his own. (Note: This is specific to this BASIC (BASIC-E), not to all DEC-like BASICs.)

Summary

We have looked at two different ways of using strings in BASIC, both are common enough to have a following, but the most useful one is the one on your computer. Which is better? It's not for me to know; however, I have used both long enough to know that strings make a program really fun to use—even if it's a business program. That is because we talk in strings, not numbers. Like other computer users, I have braved strings in FORTRAN (which has no strings) and thrilled to a real string language like SNOBOL (running on a 360/65 in 250K). You use what you have! And hope someone's coming along with something better. Until then, keep on coding! ■

References

1. *Data General Extended BASIC User's Manual,* Rev 6, Feb. 1975.
2. *DEC PDP-11 BASIC-PLUS Language Manual,* July 1975.
3. *Altair BASIC Reference Manual,* 1975.
4. *Tymshare BASIC Tycom-X Manual,* March 1973.
5. *CTSTS BASIC Reference* (INFONET, Computer Sciences Corp.), May 1974.
6. *Timeshare BASIC/2000 Level F Reference Manual* (Hewlett-Packard), Feb. 1975.
7. *North Star BASIC Version 6 Manual,* Feb. 1977.
8. *Personal Notes from GE-635 Mark I Timesharing,* Oct. 1968.

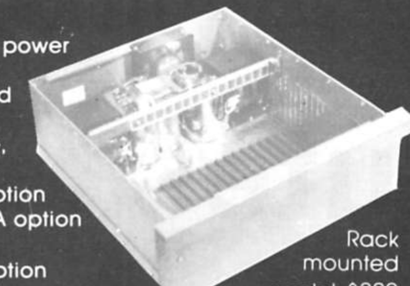

```
**** NAMES ****
SALLY JONES
SAM SMITH
JOE SMITH
TIM CAMBELL
ED HILL
STEVE MOODY
ROGER HEAD
SHIRLEY JONES
ISSAC DEAR
RICH KING

**** SORTED NAMES ****
CAMBELL, TIM
DEAR, ISSAC
HEAD, ROGER
HILL, ED
JONES, SALLY
JONES, SHIRLEY
KING, RICH
MOODY, STEVE
SMITH, JOE
SMITH, SAM
READY
```

Fig. 1. List sort.

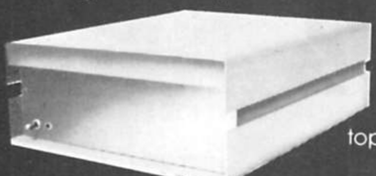
S-100 MAINFRAME $200

S-100 MAINFRAME $200

- NOT A KIT
- 8v@15A, ±16v@3A power
- Rack mountable
- 15 slot motherboard
- Card cage
- Fan, line cord, fuse, switch, EMI filter
- Desk top version option
- 8v@30A, ±16v@10A option
- SS-50 bus option
- voltage monitor option



Rack mounted model $200



Desk top model $235

Write or call for a copy of our detailed brochure which includes our application note BUILDING CHEAP COMPUTERS.

INTEGRAND

8474 Ave. 296 • Visalia, CA 93277 • (209) 733-9288

We accept BankAmericard/Visa and Master Charge

NORTH STAR

DISK ASSEMBLER and DISK EDITOR

Both programs read and write disk files; file size not limited by memory. Assembler will assemble up to ten source files at a time; permits modular programming with programs easily relocated by reassembling at the desired address. Editor does not use line numbers; it searches for strings. Lines may be inserted, deleted and displayed. Large disk source files allow programs to be fully commented.

ASSEMBLER/EDITOR on disk with users manual...$30

COMPUTER SYSTEMS DESIGN

1611 E. Central Wichita, KS 67214

DEALERS INQUIRIES INVITED

C46

YOUR BEST BUYS ARE AT

Computer Corner OF NEW JERSEY

C64

240 WANAQUE AVE., POMPTON LAKES, N.J. 07442

(201) 835-7080

Telex - 130-376 answer back AMCALL POMT

micro~mini computer center

STORE HOURS: MON.-FRI. 6:30 p.m. to 10 p.m. SAT. 10 a.m. to 6 p.m.

AUTHORIZED DEALER FOR:

• Southwest Technical Products Corp. • Seals Electronics, Inc. • Jim-Pak Electronic Components • PolyMorphic Systems • Micro-Term • Ohio Scientific • Technico, Inc. • Hayden Books • Kim-1 • Sanyo

LOOK AT THESE PRICES!

Only at Computer Corner of N.J.

TECHNICO TMS 9900 - 16 Bit Processor Super Starter System T-9900-SS-U

Regularly $299 *NOW* $269⁹⁵

Also the New **Netronics ELF II, RCA COSMAC** microprocessor, minicomputer for only **$139.95** wired & tested!

—— VERBATIM Removable Magnetic Storage Media ——

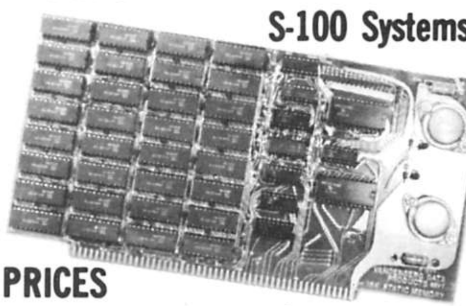
Minidiskettes	MD525-01 MD525-10 MD525-16	1-9 4.50	10-25 4.25	26-100 3.75
Standard Size Diskettes	FD34-1000 FD32-1000 FD65-1000	1-9 5.50	10-25 5.25	26-100 4.75

TO ORDER:

Send check or money order and include $1.00 for shipping, 1.00 (optional) for insurance, and please include 5% sales tax if you are a New Jersey state resident. Phone orders accepted with Mastercharge or Visa.

16K Static RAM

S-100 Systems



NEW LOW PRICES

$330 Kit **$365 Assembled**

- Very Low Power—650MA+5V; 90MA+12V; 16MA-5V
- Applications Notes—6800 and 6502 Sys.
- Low-profile sockets for all chips
- Solder mask; silk screen; plated through holes
- Each 4K addressable to any 4K boundary
- Fully buffered S-100 bus—gold-plated contacts
- NEC μPD 410 D memories

COD, Master Charge, B of A, Visa Accepted . . . . Orders shipped prepaid. California residents add 6% sales tax.

VANDENBERG DATA PRODUCTS

PO BOX 2507 SANTA MARIA, CALIFORNIA 93454 805-937-7951

V14