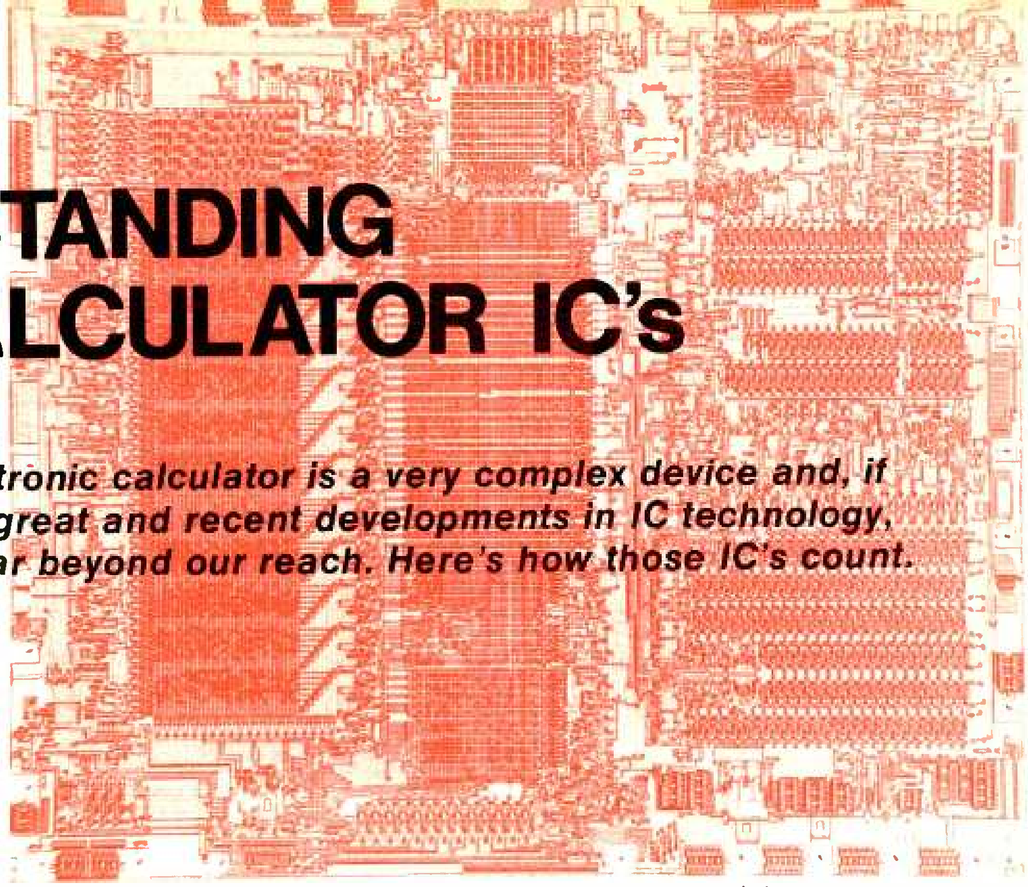
UNDERSTANDING CALCULATOR IC's



Basically, the electronic calculator is a very complex device and, if it were not for the great and recent developments in IC technology, it would be priced far beyond our reach. Here's how those IC's count.

by DON LANCASTER

MOST LOWER PRICED CALCULATOR SYSTEMS CONSIST OF very few parts—a display and driver assembly, a keyboard, a battery and case, a clocking system and decimal point and constant selectors, and, finally, a single integrated circuit that does all the work. A block diagram of a typical calculator is in Fig. 1. Numbers and program commands entered by the keyboard are carried out and used by the integrated circuit, and answers are then routed to the display.

Since the integrated circuit does everything, our overall system complexity changes very little if we add squares, square roots, percentages, metric conversion, etc. . . . All that happens is that the inner workings of the integrated circuit get slightly more complex, and perhaps a key or two is added. The way to understand these circuits is to understand what the IC does and how it works. Let's take a closer look.

Some basic operating principles

If we had no worries about total calculation time, supply power, circuit complexity, or the number of pins and interconnections on the package, there'd be a lot of different possible ways to do the job. But, when we take all these restrictions into account and at the same time aim at circuitry that *eventually* will sell for \$19.95 or even \$9.95, there's only one really good route to do the job. While best current pricing is still \$29.95 and up, the inner workings and mechanical complexity of a 4-function calculator is far less than a \$4 transistor radio.

What is the best route to do the job? It's based on several circuit principles:

- Serial by digit arithmetic
- Repetitive use of a simple, unsophisticated arithmetic unit to do complex functions
- Use of a multiplexed display
- Use of a scanning keyboard
- Use of a dynamic shift register *stack* to store inputs, calculation products, and answers.

Let's look at these one at a time, starting with the simpler concepts, and then going on to put the whole thing together as a working system.

Multiplexed displays

Our display typically consists of a group of light emitting diodes (LED's) or an arrangement of neon display characters, although some calculators also use miniature fluorescent display tubes or liquid crystal readouts. Regardless of the method we use, we have to select a means of driving the display that has a minimum need for power, interconnections, and storage restrictions.

If we were to display all the digits all the time, we'd need at least 80 leads for a 7-segment, 10-place display. This is clearly inefficient.

Instead, *we only display one digit at a time,* but we *sequentially* change *which* digit we display fast enough that the eye averages everything out into a continuous process. Each digit is lit to many times "normal" brightness for a fraction of the total time. The final result is a digit that is *apparently* continuously lit to normal brightness. This circuit trick is called *display multiplexing*.

Multiplexing is shown in Fig. 2. Most calculator systems internally use a 4-bit BCD (Binary Coded Decimal) code. To drive the popular 7-segment displays, the chip internally converts the 4 BCD bits into the proper 7-segment patterns.

The output of the 7-segment converter goes to *all* the digits being displayed in parallel. Now, the trick is that we use a one-of-N decoder to provide supply power or *digit line power* to only *one* entire numeral at a time. Thus, while *all* the digits know what the numeral is to be, only *one* of them receives supply power at any single instant, and only one of them lights at a time. We bring the digits information out one digit at a time, and at the same instant, we step and decide which digit line gets power. The result is a sequence of digits being lit. The sequence repeats so fast that you see everything as a continuous display.

One very handy circuit-saving trick the calculator people use is that they sequence the digits at the same rate they do the calculations and in the same order—thus our digit sequencing is essentially "free," as the numbers have to go around anyhow when a calculation is being made. When a calculation is *NOT* being made, the numbers still go around, but the internal arithmetic unit (more on this in a bit) goes into a *do-nothing* mode and does not change the contents of the *display or answer register* that is storing the number being output.

Multiplexing needs a display system that has internal diodes, thresholds, or nonlinearities. If the system was perfectly linear, you would get *sneak paths* through series combinations of supposedly off segments, causing ghosting and other problems. Light emitting diode displays are inherently diodes, and eliminate the problem, as do the nonlinearities associated with fluorescent displays. Panaplex and Sperry neon displays have a well-defined threshold that also eliminates the problem. Liquid-crystal readouts do not inherently multiplex very well, and for this reason, very few calculators use the otherwise ideal liquid crystal display systems.

Driving a display

Ideally, we'd like to come directly out of our calculator IC, and go directly to the display without needing any interface circuitry at all.

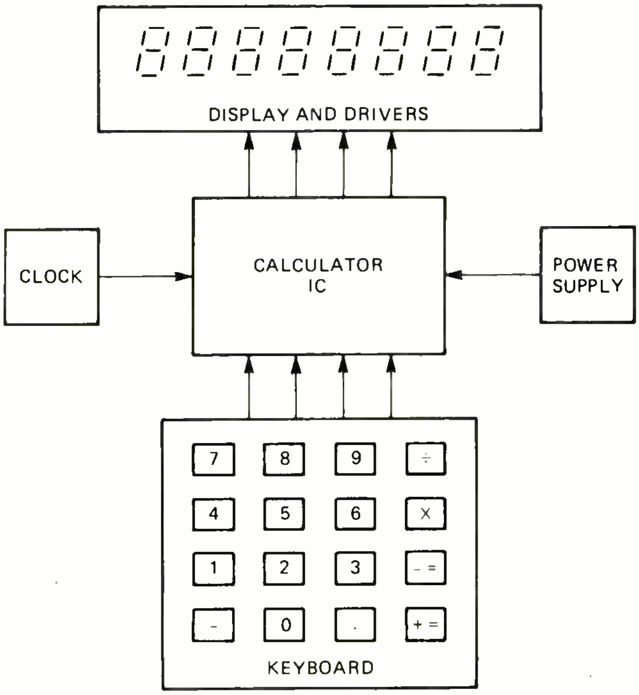


FIG. 1—CALCULATOR BLOCK DIAGRAM. Keyboard generates 4-bit BCD numerals. The clock — a multivibrator — controls operational sequence.

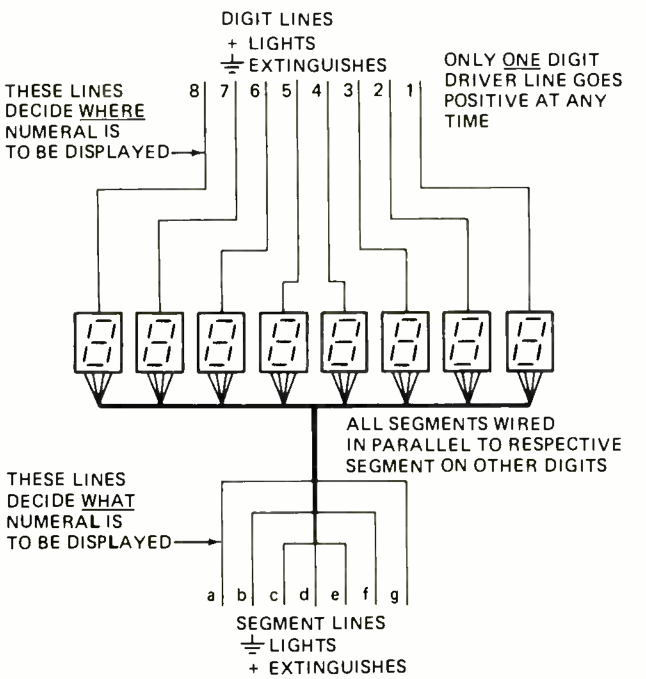


FIG. 2—THE SEVEN-SEGMENT READOUTS ARE MULTIPLEXED to reduce circuit interconnections and instantaneous power drain.

Unfortunately, every display system in use today has some interface restrictions, owing to needing more current or more voltage than a calculator IC can provide. Some typical interface circuits are shown in Fig. 3. In the case of neon-type displays (Fig. 3-a), drive voltage is usually the problem. Most neon systems need considerably more drive voltage than the calculator IC's can provide. For LED's, current is the problem, for each individual segment needs a few mils, while the digit driver lines may need several *hundred* milliamperes of drive, since a digit driver may have to drive up to 7 segments at once, and since multiplexing magnifies the peak-to-average current ratio additionally by a bunch.

Figure 3-b shows how two driver integrated circuits (the Texas Instruments 75491 and 75492 being typical) are used to drive a LED-type display. Special high-voltage integrated circuits are available for neon-type displays, or discrete circuitry consisting of Darlington connected transistors and resistors may be used.

Obviously, the $9.95 calculator won't be able to afford any inter-

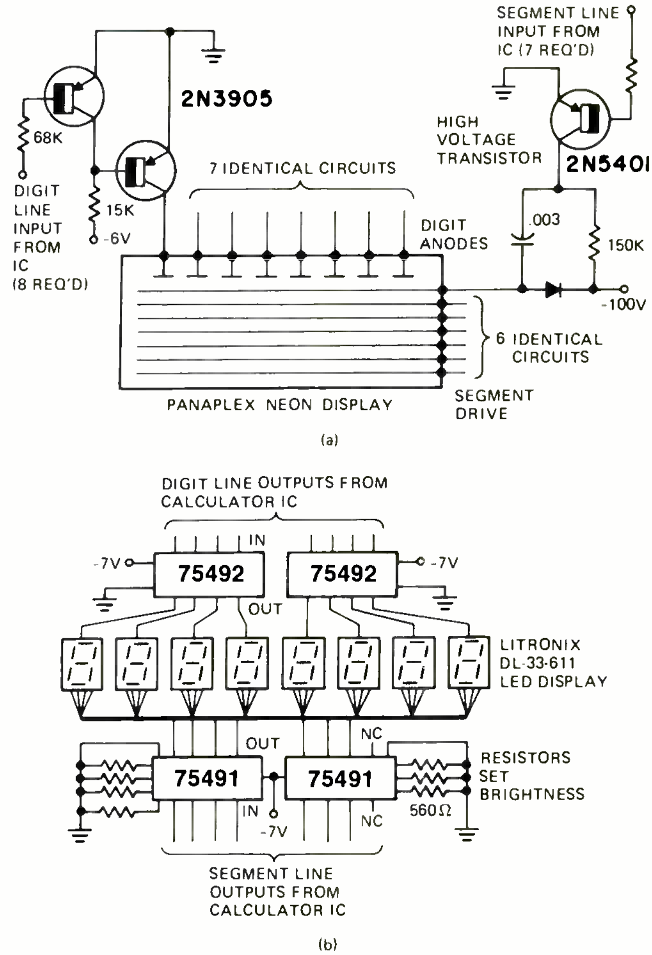


FIG. 3—DISPLAY DRIVERS ARE NEEDED when the calculator IC cannot handle the current or the voltage required by the readout.

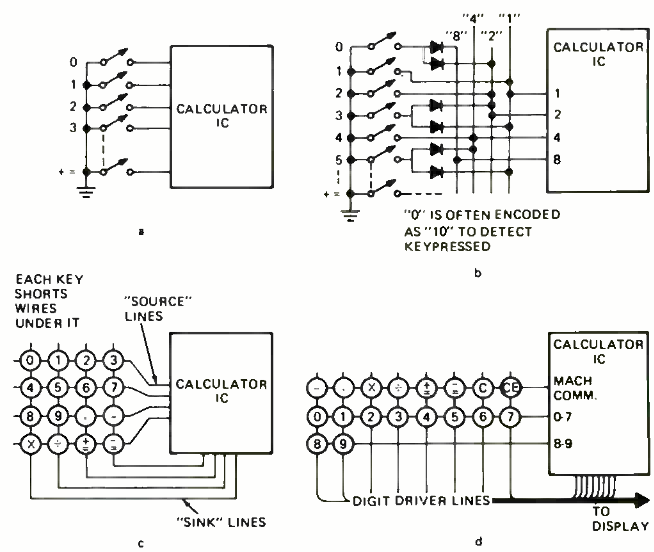


FIG. 4—KEYBOARDS MUST BE SIMPLE to minimize the number of lines and external parts. Matrix (c) and scanning types (d) are best.

face circuitry. Newer calculator IC's such as the *General Instruments* CZL-550 can directly drive some LED display segments, and a companion IC is available for digit line driving. At the same time, LED systems are becoming more efficient, and reducing their current needs to levels that are compatible with calculator IC technology. Liquid-crystal displays, with their essentially zero current consumption and low-voltage operation, should eventually dominate the calculator display market, *if and when* practical multiplexing schemes are worked out and a few other production and life problems are worked out.

With today's chips and circuitry, you still must use some sort of interface circuitry, either in the form of additional IC's, or Darlington connected transistors and resistors. The interface circuitry obviously must meet both the needs of the IC and the display, taking into account the data sheet values of each device.

Scanning keyboards

Our keyboard data entry system must use the simplest possible key switches, minimum external encoding components, and a minimum number of package pins. How can we do all this at once? Several keyboard systems we could use are in Fig. 4.

Figure 4-a shows about the worst we could do. Here, we have one common lead and one output lead for each individual key. If we had 16 keys, this would take 17 leads. We can cut down on the number of leads with a diode encoding network as in Fig. 4-b by encoding 1-of-16 to 1-2-4-8 4-line binary. This takes a bunch of diodes, but saves us on package pins. In a system of this type, we might make a "0" the equivalent of a binary "10," or 1010. This way, our "0000" state can be a do-nothing state, and any other combination indicates a key-pressed command. Thus, binary 1 through 10 would be the numbers, and binary 11 through 15 would be the add, subtract, multiply, divide, and equals commands.

We can go to the keyboard matrix of Fig. 4-c to eliminate the diodes. Here we arrange our keys in a 4 x 4 array. There are four lines *from* the calculator that *source* current; there are four lines *to* the calculator that *sink* current. Press any one key, and one of the source lines gets shorted to one of the sink lines. There are 16 possible combinations of source-to-sink shorts that may be uniquely internally recognized as a numeral or a machine command.

This eliminates the diodes, but we still need 8 input leads. Can we do any better? Remember that we already have 8, 10, or 12-digit driver lines going to our multiplexed display. If we use these in our matrix as in Fig. 4-d, we end up with a *scanning keyboard* that only needs *two* new leads for a 16-key system, and only *three* new leads for up to 24 keys if there are eight digit drivers. Pressing any one key connects one of the digit driver lines to one of the keyboard input lines. The internal circuitry recognizes the time slot of digit driver pulse, and enters the appropriate numeral or machine command. While quite a bit of internal circuitry is needed, the same circuitry also provides debouncing, minimizes the package pins needed, and eliminates any need for diodes or other encoding components.

Any given calculator chip can only work with one of the 4 systems of keyboard entry shown—the newer the chip, the more likely it will use the scanning configuration of Fig. 4-d since it is the best in terms of pins and simplified external circuitry.

Inside the chip

So, what goes on *inside* the calculator chip? There are several important circuit areas, as the block diagram on Fig. 5 shows us. These essential parts include the *memory stack*, where numbers are stored; the *arithmetic unit*, where *simple* arithmetic operations are carried out; the *microprogram*, where the simple arithmetic unit is told what to do over and over again and in what sequence; the *keyboard interpreter*, where numbers are debounced and entered and machine commands (add, subtract, multiply, equals, etc. . . .) are sorted out and used to sequence the microprogram; finally some *housekeeping* functions, such as decimal point circuitry, output BCD to 7-segment conversion, etc. . . .

There are two basically different types of memory in a calculator IC. The memory stack contains the numbers and changes from time to time, as we enter new numbers or as changes are made as a calculation is being carried out. This changeable memory is called a read-write or Random Access Memory (RAM). On the other hand, the microprogram sequencing information need never be changed, for once we teach the calculator how to sequentially perform a calculation, we henceforth and evermore want it to do the same thing. So, the microprogram is in a *fixed* data storage mode, and is called a *read-only* memory. Let's look at the memory stack first.

Remember that our multiplexed display needs something that goes round and round, sequentially putting out digits in the proper order so they can be displayed. A memory that does this is called a *shift register*, and the shift register is arranged to normally *re-circulate* its numbers by connecting output to input via a logic circuit. Calculator shift registers usually are dynamic ones; they have to be kept moving at all times or the data will be lost. Arithmetic is usually done in the 4-bit BCD code. This takes four bits per numeral, so we really have four separate but identically clocked shift registers to march the numerals around, one at a time. Such a system is called a *parallel by numeral, serial by digit* arrangement. A typical memory from a memory stack is in Fig. 6. The length of the shift register used is determined by the number of digits to be displayed, plus a sign digit, plus some possible locations for overflow and underflow digits. Regardless of its length, at any instant, all four bits of a digit appear at the output of a stack memory.

The memory is shifted or advanced one stage at a time, usually by an internally derived clock one-fourth the frequency of the system clock you provide to run the IC. It takes nine or more internal clockings to exactly once turn over the memory, depending on the number of stages in the shift register. The turn-over time is called the *machine cycle time*, and a calculator with a 25-kHz-or-so system clock would end up with a machine cycle time around 2 milliseconds or so.

If we connect a stack memory to a BCD, to-7-segment decoder, and then connect this decoder to our display, the number stored in that portion of the memory stack will be displayed.

One stack memory can only store one number. It takes more memory than that to do the job. We need an *answer* memory and a *keyboard* memory at the very least for simple addition and subtraction. If we are also to multiply and divide, we also need an *arithmetic* or *operand* memory, and, as an option, if we are to store a constant, we would need a fourth *constant* memory. So, there are three or four separate shift register memories in our memory stack, one for keyboard, one for answer, one for arithmetic, and an optional one

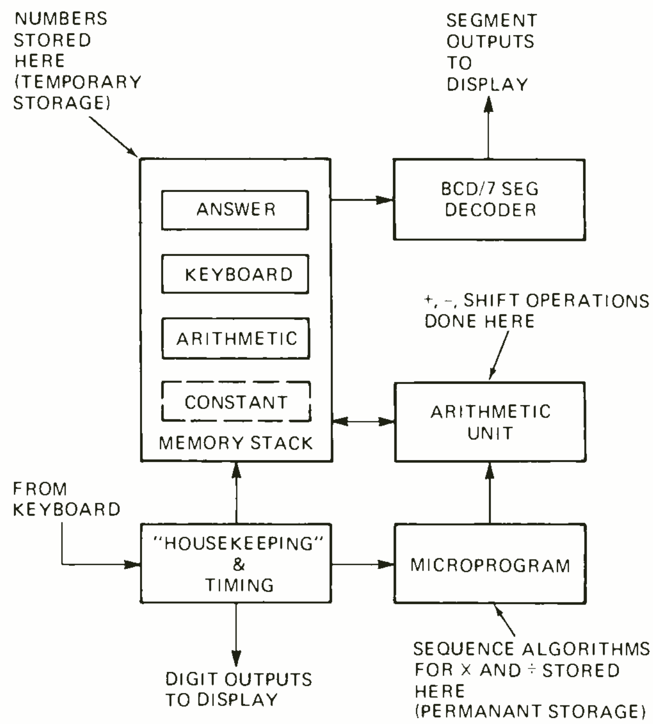


FIG. 5—INSIDE THE CALCULATOR IC. The "keyboard" section reads and sorts out input signals from the keyboard.

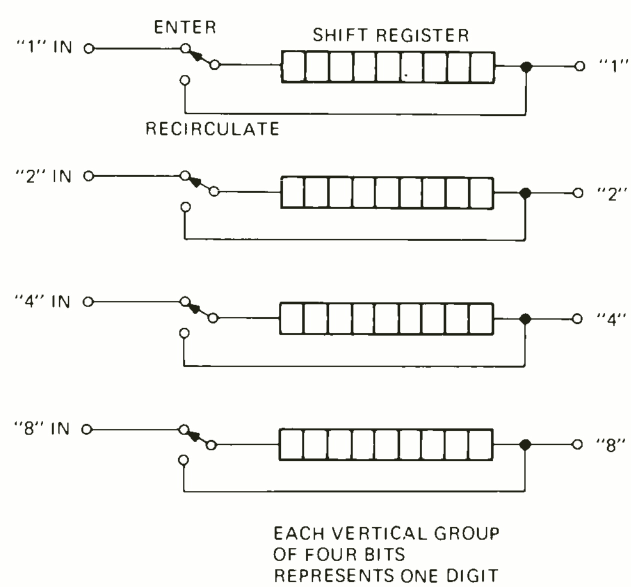


FIG. 6—TYPICAL MEMORY in memory stack. Length of shift register depends on number of numerals and sign digits to be shown.

SORRY - PAGE 41 IS NOT YET AVAILABLE

UNDERSTANDING CALCULATOR IC'S

(continued from page 41)

library or *Electronic Design*, 50 Essex St. Rochelle Park, N.J. 07662.

With 500 or so machine cycles available each second we can use rather sophisticated algorithms and still come up with an apparently "instant" answer to a tough problem. Minicomputers and regular computers use parallel computational systems with much faster machine cycle times of fractions of a microsecond. These are much more expensive and are not needed for one-at-a-time arithmetic operations.

The rest of our calculator chip takes care of the "housekeeping", cycling the microprogram in the proper order, taking care of constants and decimal points, accepting information into the keyboard register, routing the display to a BCD to seven segment converter and the proper memory in the stack, and so on.

What's available?

There are dozens of different calculator chips available today, both as new and surplus items. Some of these are now as low as $5 each surplus, and quality new units in production quantities are pushing a $4 figure. A few of the more common calculator IC's are on page 41. A list of some of the manufacturers is shown below.

A Few Calculator IC Manufacturers

AMERICAN MICROSYSTEMS
3800 Homestead Road
Santa Clara, California, 95051

CAL-TEX SEMICONDUCTOR INC
3090 Alfred Street
Santa Clara, California 95050

GENERAL INSTRUMENTS
600 West Johns Street
Hicksville, New York, 11802

INTEL CORPORATION
3065 Bowers Avenue
Santa Clara, California, 95051

MOS TECHNOLOGY
Valley Forge Center
Norristown, Penna, 19401

MOSTEK
1215 West Crosby Road
Carrollton, Texas, 75006

TEXAS INSTRUMENTS
Box 5012
Dallas, Texas, 75222

As with any IC, if you are building or experimenting with these, be sure to have all data sheets and applications notes on hand before you start, along with whatever other information you can possible get—and read everything carefully.

Broken down into its component parts, there's nothing really very fancy or exotic about a calculator—except, of course, for the incredible amount of engineering and expertise that goes into successful chip design. Calculator IC's are now cheap enough that you should be able to do much more with them than simple four functions arithmetic, particularly if you add *your own external* microprogramming, entering in parallel with the key commands. What applications can you think of? **R-E**