VOL. 48, No. 3 RADIO-ELECTRONICS

BUILD A LOGIC PROBE FOR $16

$1.00 ■ MAR. 1977

Radio-Electronics

THE MAGAZINE FOR NEW IDEAS IN ELECTRONICS

a look at
NEW HI-FI GEAR
for tomorrow's audiophile

get rid of it—
RF INTERFERENCE
in your hi-fi

build a portable
ACTION FOOTBALL GAME
using IC's

easy to build
DIGITAL CLOCKS
...dules

...ked-loop
S... SIZERS
...annel CB

LEN FELDMAN REPORTS—
Binaural and Biphonic Sound

Sitting in this new JVC experimental 4-channel biphonic chair, our Contributing High-Fidelity Editor becomes one of the first to experience listening to binaural sound without wearing headphones

★ ...Multimeters ★ Komputer Korner
★ ...r's Service Clinic
★ State Of Solid State ★ Equipment Reports
★ Lab Tested Reports on Nakamichi 610 Preamp and Garrard GT-55 Turntable

GERNSBACK PUBLICATION

KOMPUTER KORNER

Stacks—what they are and how they're used

TIM BARRY

MICROCOMPUTER USERS THESE DAYS ARE CONstantly bombarded with a bewildering supply of new jargon. In addition to the normal proliferation of new hardware terms, we now have to cope with words from software design, systems analysis, and a whole herd of other less well defined disciplines. One of the most commonly used (and abused) terms these days is *stack*. The hardware represented by this picturesque term is often endowed with rather mysterious qualities. Vague utterances about "pushing" and "popping" blend together with questions about "balancing" and "nesting" to create an ample atmosphere of confusion. In this article we will look at two principal types of stacks and how they operate. In doing this we will hopefully dispel some of the myths surrounding these extremely versatile devices.

What is a stack?

In the most general terms, a stack can be considered to be any serial storage system. A stack will have an input end and an output end. All data placed into the stack must pass one element at a time through the input end of the stack. Once an individual data element is in the stack, it can only be accessed by removing preceding or succeeding data elements until it reaches the output end of the stack. The order in which data is placed into and removed from the stack differentiates between two different types. Figure 1 illustrates the two different types.

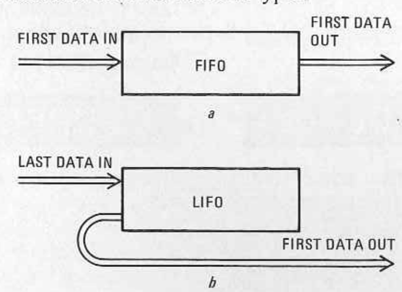

FIG 1

The first type we will discuss is probably the easiest to understand. In these days of crowded facilities and long lines, we, as people, all spend plenty of time in FIFO (*F*irst *In*, *F*irst *Out*) stacks. The data entered into the input end of a FIFO emerges from the output end in the same order in which it was entered. (See Fig. 1-a.) The principal use of FIFO's is to store arriving data for later use. The next time you are waiting in line, you can reflect on the fact that you are participating in a genuine computer buzz word.

The second type of stack is less commonly encountered in our day to day experiences. A LIFO (*L*ast *In*, *F*irst *Out*) stack returns data in the opposite order from which it was entered. (See Fig. 1-b.) This means that all the data in the LIFO must be removed before the first data element entered can be recovered. For example, consider an empty bus. Assume that each passenger who gets on the bus goes to the back and no one gets off enroute. When the bus unloads, it should behave like a LIFO, with the last passenger that entered being the first one off. LIFO's are most commonly used in computer programming to save program data and subroutine return addresses.

continued on page 24

KOMPUTER KORNER
continued from page 22

FIFO operation

FIFO stacks are most commonly encountered in computer I/O systems and data acquisition systems. They are used to match the data transfer rates between two systems. In this type of application they are often referred to as *storage buffers.* The need for these devices arises when data is transmitted in bursts which are to fast for the receiving device to process. The FIFO is used to store the data in the order in which it arrives during the entire burst. The receiver (usually a computer) can then process the data from the FIFO at its own rate. A common FIFO application of this type is found in computer disk systems. Data to be transferred to or from a disk must be transferred at a higher rate than most computers can manage. To solve this problem, a FIFO buffer is used. The disk transfers a block of data into the FIFO at high speed. The computer can then use the data at its own rate.

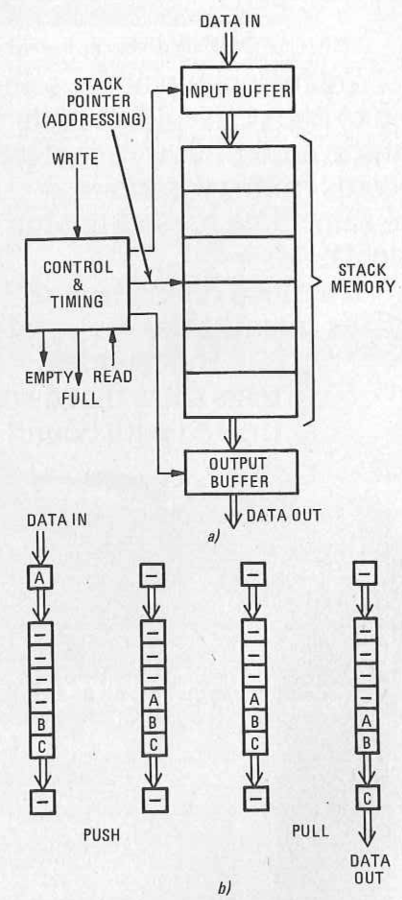

FIG 2

FIFO's can be implemented in either hardware or software. Hardware FIFO's are actually available as LSI integrated circuits from several semiconductor manufacturers. They contain data buffers, registers, a memory, and all required control logic. The block diagram of a typical FIFO (simplified to 6 locations) is shown in Fig. 2-a. In operation, the FIFO accepts data into the input buffer. It then *pushes* the data down in memory until it rests in the first empty location in the memory. When data is read

continued on page 26

KOMPUTER KORNER
continued from page 24

from the FIFO, the first data element in the memory is *pulled* into the output buffer. All other data in the memory is then moved down one. If the FIFO becomes full or empty, it sets flags to indicate these conditions. A graphical illustration of a FIFO operation is shown in Figure 2-b.

A FIFO can be implemented using software instead of hardware. In this case subroutines are used to perform the push and pull operations. The flow chart for the push subroutine is shown in Fig. 3-a and the pull subroutine is shown in Fig. 3-b. In actual use, these two programs will both share a common block of the computer's main memory for the FIFO stack. They will keep track of where the data in the FIFO is by using a *stack pointer.* A stack pointer always indicates the memory address of the most recent data element entered into the stack. (This is true of both FIFO's and LIFO's.)

When the push subroutine is executed, the stack pointer is incremented by one and the new value returned to the memory. The data passed to the subroutine is then transferred into the stack memory at the address now indicated by the stack pointer. For a pull operation, the stack pointer is first tested to see if the stack is empty. If it is, an error flag is set and the subroutine returns. If it is not empty, the data at the first memory address of the stack is obtained. The stack pointer is then decremented by one and re-stored in the memory. All the data in the FIFO is moved down one in the memory. The routine then returns to the calling program with the data.

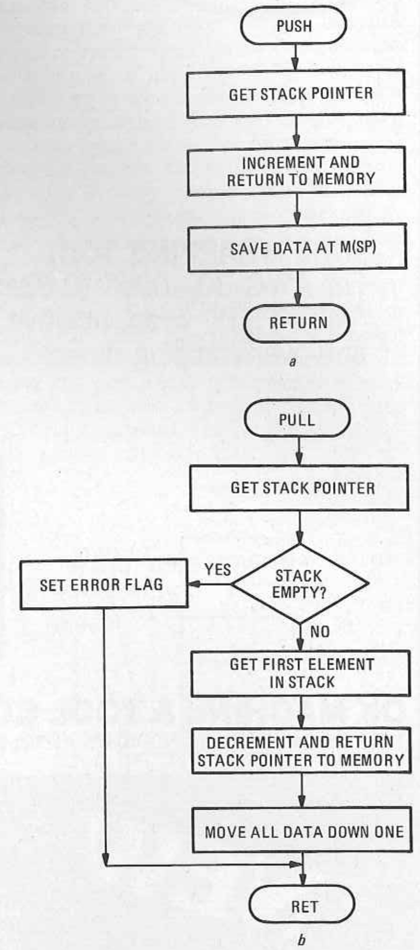

FIG 3

LIFO operation

The LIFO stack is most commonly encountered in systems programming and microcomputer subroutine systems. In systems programming (particularly language processors), the LIFO is used to store data and operands during the evaluation of arithmetic and algebraic expressions. In microcomputer subroutine systems, the LIFO is used to hold program data and subroutine return addresses.

The hardware required to implement a LIFO consists of the same basic hardware we saw used in the FIFO. It is simply connected together in a different configuration. (See Fig. 4-a.) In operation, the data is received into the input buffer just as before. However, all the other data in the LIFO is now pushed down one to make room at the top for the new element. For an output operation, the top element of the stack is *popped* into the output register and all the other data is then moved up in the memory. These LIFO operations are illustrated in Fig. 4-b.

Implementing the LIFO can also be done in software. The flowchart for the push subroutine is shown in Fig. 5-a and the pop subroutine is shown in Fig. 5-b. As with our software FIFO, the software LIFO routines will share a common memory area and stack pointer. When the push subroutine is called,
continued on page 32

KOMPUTER KORNER

continued from page 26

the stack pointer is incremented by one and the new value returned to the memory. The data passed to the subroutine is then transferred into memory at the address indicated by the stack pointer. For a pop operation, the stack pointer is first tested to see if the stack

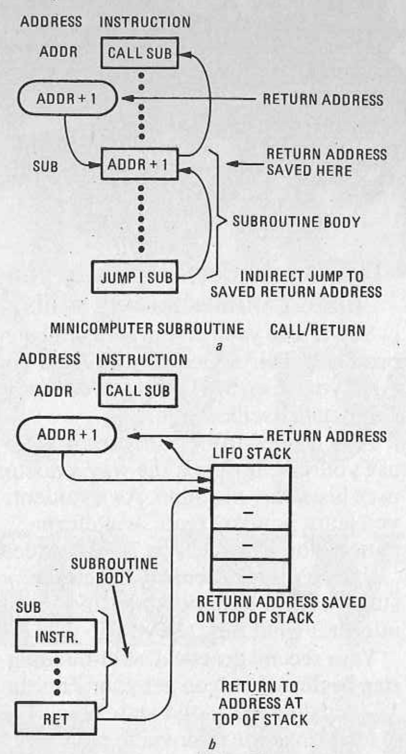

FIG 6

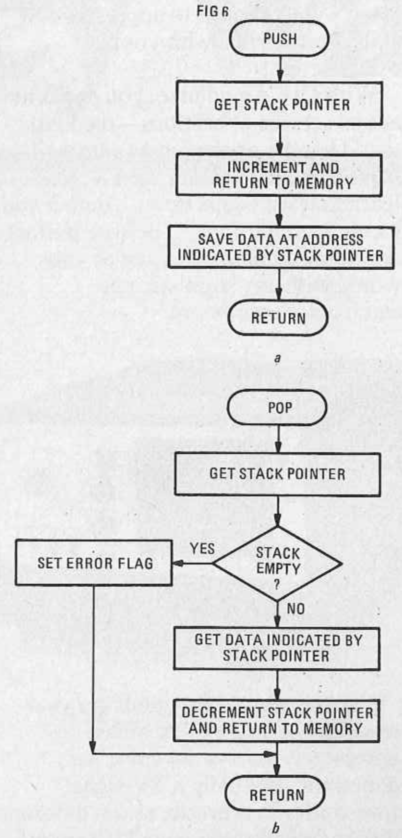

FIG 5

is empty. If it is, an error indicator flag is set and the routine returns. If the stack is not empty, the data at the address indicated by the stack pointer is obtained. The stack pointer is then decremented by one and the new value stored in the memory. The routine then returns to the calling program with the data.

Stacks and microcomputers

As mentioned earlier, many microcomputers use stacks to implement their subroutine systems. This arose because of the unsuitability of the normal minicomputer way of saving return addresses. When a subroutine is called, most minicomputers save the return address of the calling program in the top location of the subroutine called. A return can then be executed by performing an indirect jump to the first location in the subroutine. (See Fig. 6-a.) This works great in systems where the subroutines are located in read/write memory. It's not so hot when the subroutines are to be located in read only memory. Since most microcomputers make extensive use of ROM, something had to be done. Enter the LIFO.

When a LIFO is used for the subroutine structure everytime a subroutine is called, its return address is pushed onto the stack. When a subroutine return is executed, the top address in the stack is popped into the program counter, thus transferring control to that location. (See Fig. 6-b.) Now this means that all subroutines must be returned in the

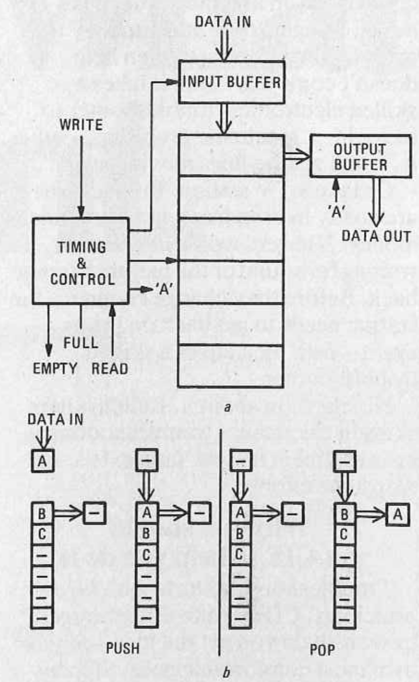

FIG 4

opposite order in which they were called if proper program operation is to be maintained. The number of subroutine calls executed before a return is executed is called *nesting*. Thus the phrase, "My program is nested five deep", means that the program has called five subroutines before the first return has been executed.

Microcomputer designers took two basic approaches to implementing the LIFO stack for return addresses. The first way was to build the LIFO in as part of the CPU hardware. This method required no special

continued on page 86

KOMPUTER KORNER

continued from page 32

user operations and solved most problems associated with the subroutine system. The main drawback of this method was imposed by the semiconductor processing technology. With the stack on the same chip as the processor, there wasn't a lot of room left. This meant that the number of elements in the LIFO had to be limited. The result was a limit on the amount of program nesting that could be performed without filling the stack and losing return addresses. On most processors the user was limited to seven unreturned subroutine calls. This limitation also meant that stack usage had to be limited to subroutine return addresses alone. This denied the user access to many of the other features a stack can provide.

To solve these two problems, some manufacturers decided to implement the LIFO as part of the system main memory. They provided the stack pointer and the automatic increment/decrement hardware. The user then supplied the address of the memory block to be used as the LIFO. This is accomplished by loading the top address of the selected memory block into the stack pointer. From then on the stack functions automatically.

This certainly solved the nesting problem. Most systems have far more memory than a properly functioning program will ever need for subroutine nesting. It also makes the stack available for other uses. This allows you to use the stack to save registers, pass data to and from subroutines, and lots of other useful functions. However, there is no such thing as a free lunch. If you are going to manipulate data in the stack you have to *balance* the stack pointer.

Balancing the stack pointer simply means that it must be pointing to the correct return address when a subroutine return is executed. Failure to do this can result in your program accidentally using the data you meant to pass back in the stack as the return address. This type of error can result in some really interesting program execution. Balancing the stack requires that you pay close attention to the order in which data is entered and removed from the stack. Under normal operations, the data must always be removed in the opposite order in which it was entered. You must also be certain that no programs accidentally write data into the area you have reserved as the stack. It is also a good idea to make sure the stack doesn't grow too large and encroach on other program storage. The easiest way to avoid most of these problems is to assign the stack to the top 100 bytes of your system memory and leave it alone. This will probably be far more stack than you ever need and it will save you a lot of time you would otherwise have to spend computing exact stack usage.

Summary

Stacks provide you with a convenient way to solve many design problems. The FIFO and LIFO offer different characteristics for use in different applications. They can both save you much time and make it easier to implement a variety of system functions. If your computer uses a memory LIFO stack for subroutine return addresses, with a little practice you will discover ways to use it to make your programs more efficient. Whether hardware or software, the stack is a useful new tool for the designer. **R-E**

1977 CB sales will exceed all previous years combined

More CB radios will be bought in the United States in 1977 than in all of CB's previous 28 years added together, says John Sodolski, vice president of the communications division of the Electronic Industries Association (EIA). He expects sales to approach the ten million mark for 1977, and estimates that retail sales of CB radios, antennas and accessories should top $2 billion for the year.

Only about three million CB radios were sold between 1958—when the FCC allocated 23 channels for Citizens radio—and 1973, when the sudden upsurge began. Sales exceeded a million in 1973, then doubled each succeeding year, hitting nearly five million in 1975.

Unlike some industry predictors, Mr. Sodolski believes that 23-channel radios will continue to be popular, especially during the earlier part of 1977, before the supply of 40-channel sets catches up with the demand. Favorable pricing and the realization that the 23-channel radio satisfies the needs of a majority of the people in many parts of the country are the important factors that will keep the 23-channel sets moving, he says. **R-E**

EQUIPMENT REPORTS

continued from page 34

hold the penlight? You don't; you hold it between your teeth just as you always did.)

The *model 175* is powered by self-contained rechargeable batteries. All of the very complex "works" are on only two boards, one for the logic, and the other for the display. A proprietary LSI/MOS IC performs all of the logic functions required by the A/D converter. Due to the extensive use of MOS circuitry, the total power consumption is less than 1.0 watt when used with the AC powered battery charger, and 0.6 watt on battery alone. The battery is good for up to 6 hours of normal operation with a full charge, and it can be recharged overnight; 12 hours. When the battery needs charging, the decimal point of the display *blinks* continuously!

The inherent high accuracy of the digital multimeter is taken full advantage of here. On DC volts, the accuracy is 0.1%. On AC volts between 50 Hz and 500 Hz, the accuracy 0.4%. To verify this, a complete set of the final test calibration readings is packed with every instrument. Specification limits are given and the actual test reading logged. On the 1,000 volt range of the one we reviewed, for example, the spec was +998 to +1002 volts. Anywhere between these limits, OK. The actual reading was 1,000 volts. Each instrument is given a burn-in test for 8 hours, and the calibration is then rechecked.

Everything else is automatic. The decimal point is automatically positioned correctly. On all DC measurements the polarity indicator is automatic. As is customary in DMM's, the resistance is read out in 1000 ohms, except for the very lowest range, 100 ohms. Overrange greater than 100% is indicated by a blanked display, leaving only the decimal point and polarity indicator lit.

Overload protection is provided as mentioned. If you accidentally go too far while reading AC or DC currents, they have provided a 2.0-ampere fuse located inside the handle of the red test prod! Just unscrew the black tip and the fuse pops out. Be sure to use only fast-blow type fuses for replacement. A spare fuse and push-on clips are also provided. The whole thing—meter, charger, test leads and all—can be stowed in a handy zipper carrying case.

There is a small pull-out stand on the underside of the case, to raise the front panel to an easier viewing angle. This folds for storage. A very detailed instruction manual comes with each instrument. This gives not only the correct method of operation, but a circuit description, parts lists, calibration data and a schematic with parts layout.

A very useful little instrument, and very reasonably priced for one with this kind of accuracy and reliability. **R-E**

Heath IP2718 Tri-Power Supply

THE HEATH COMPANY HAS INTRODUCED QUITE a range of power supplies for bench and experimental use. A typical example of these is their new *IP-2718* "Tri-Power" supply. This supply is intended for general experimental work in either analogue or digital circuitry. TTL and similar devices use a 5-volt DC power supply. The *IP-2718* has one;

CIRCLE 50 ON FREE INFORMATION CARD

regulated by an IC voltage regulator that also provides internal protection against overload, short-circuits and high-temperature conditions. Its current or voltage can be read on the panel meter.

For analogue circuits requiring a positive and negative DC voltage, there are two completely isolated 20-volt DC supplies. Each of these has a maximum output of 500 mA and can be continuously varied from 0 to 20 volts by the front-panel controls. Current or voltage in each supply can be read on the meter. These, too, are tightly regulated by transistor voltage regulators. The regulation is specified as less than 0.1% variation from full load to no load. Filtering is good; the maximum ripple level is only .005 V (5.0 millivolts RMS.)

All three of the DC supplies (one 5-volt and two 20-volt supplies) are completely isolated from each other and from the instrument ground. They can be tied together in

continued on page 94