Build a Computer-Controlled Kill Switch

BY JOHN YACONO and MARC SPIWAK

You need not hang around your computer waiting for a long job to finish.

For those of you who are fortunate enough to have your computer do work for you unattended, it really can be irritating when you have to return to it, just to turn it off. And it goes against the laws of logic to leave the computer on until you return, which could very well be the next day. We have a gadget that will eliminate that predicament: our computer-controlled *Kill Switch*. Not only can the device be used to turn off the computer that's controlling it, but it can also be used to turn off, by computer control, anything else that plugs into an outlet.

The Switch can automatically turn off a computer and a peripheral after handling a FAX or modem communication, printing a document, or allowing a lengthy program to complete (so you can begin your home-ward commute, or whatever, that much sooner). It can enable you to control AC devices via your computer, and when used with a timer program, it can turn something off according to a schedule. Although it connects to the computer's parallel port, it is designed to pass signals through to a printer and will not interfere with normal printer operation.

The Parallel Interface. Simply put, the Kill Switch does its job by comparing the data (8-bit characters) coming from your computer's parallel-printer port to a value (an 8-bit character) set by the user. When there is a match between the preset value and data from the computer, the unit shuts down both the computer and itself. To help explain how the comparison is performed, let's talk about how data appears at a parallel printer port in the first place.

Oddly enough, most parallel ports on the back of IBM-compatible computers sport a DB-25 connector. That's odd because the DB-25 connector is the standard connector used for serial interfaces. The connector is usually female (having holes instead of pins) to distinguish it from any serial connectors (which are normally female only on cabling) that may also be on the computer.

The designations of each pin on the DB-25 connector are shown in Fig. 1. The signals that occupy those pins can be broken down into four basic groups: grounds, data outputs, handshaking inputs, and handshaking outputs. In Fig. 1, grounds are denoted by circles, handshaking inputs are indicated by arrows pointing to the connector, and outputs (both for data and handshaking) have arrows pointing away from the connector. (Note that some of the signal lines have a fairly standard abbreviation shown in parenthesis.)

The grounds perform two jobs: One, they link the signal grounds of the two devices being connected so they can share a common ground to use as a signal reference. Two, since the connection between the two devices is often made via ribbon cable, the grounds (often called ground returns when discussed in this context) act as shields for the more important lines. For example, the wire connected to pin 19 on a ribbon cable would shield pin 6 from pin 7, and vice versa. That prevents D4 signals (whose function we'll get to in a bit) from capacitively affecting the D5 line, and vice versa. In quality cables not made with ribbon cable, each ground return is twisted around a signal line to form a twisted pair that provides a little shielding.

As their name implies, the data outputs transfer information from the computer to a parallel peripheral. That is done eight bits (one byte) at a time using pins 2–9. Bit D0 is considered the least-significant bit (or LSB) and D7 is the most-significant bit (or MSB). (Note that some computer manuals use designations D1–D8 instead of D0–D7.)

The bits, as well as all the other signals, are represented by standard TTL voltage levels; a signal between 2.4 and 5 volts is a high, or a binary 1, anything between 0 and 0.8 volts is a low, or binary 0. Anything between 0.8 and 2.4 volts is considered invalid data. Table 1 shows all standard IBM-compatible characters and their associated logic values.

Since a computer is much faster than any peripheral that it communicates with, it could easily transmit more data than a peripheral could handle. So peripherals use special signals to tell the computer to momentarily stop sending data when they have enough to work with. That gives the peripheral a chance to catch up and the computer can perform some other task in the meantime. Once sufficiently caught-up, the peripheral tells the computer to transmit more data and the process continues.

That computerized game of "red light, green light" is accomplished by sending signals along wires dedicated to that purpose. The process of using signals to control the flow of data is called "handshaking," so the

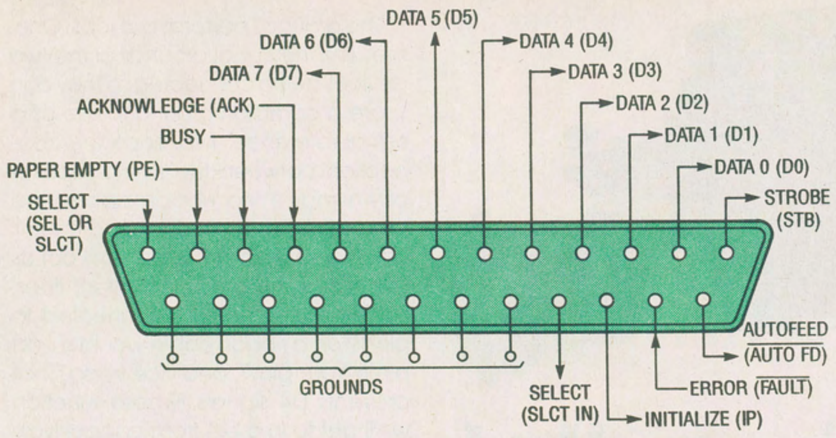


Fig. 1. The parallel port on most IBM-compatible (and some non-compatible computers) looks like this. The arrows pointing away from the DB-25 connector are outputs, the ones pointing to the connector are inputs, and the remaining pin outs (shown with circles) are grounds.

signals used for that purpose are called "handshaking signals."

The strobe, busy, and acknowledge signals are the most important handshaking signals. To help explain how they are related and control data flow, take a look at Fig. 2. There the eight data lines are shown lumped together at the top as a single band. Don't let that throw you, the value of the individual bits is not important. What is important is the time at which data undergoes a transition (represented by the crossed lines) and the time that it remains constant (the bands).

The data that is being output on lines D0–D7 starts to form at time t1, and it settles down and is ready for use by time t2. A moment later (at t3), the computer sends a momentary low-going pulse (called the strobe signal) to the peripheral to indicate that the data is ready and waiting on the data lines. After t3, the peripheral may respond in one of two ways: it can pull the busy line high until it's ready for more data, or it can wait until it has used the new data and then send a low-going acknowledge pulse to the computer when it wants more. Either response keeps the computer from proceeding until the peripheral says it's ready. After the busy line goes low or an acknowledge pulse is received, the computer will set up the data lines for the next byte, and the procedure repeats.

Along similar lines, sometimes parallel peripherals (especially printers) use dedicated wires to indicate their status. Since the status of a peripheral

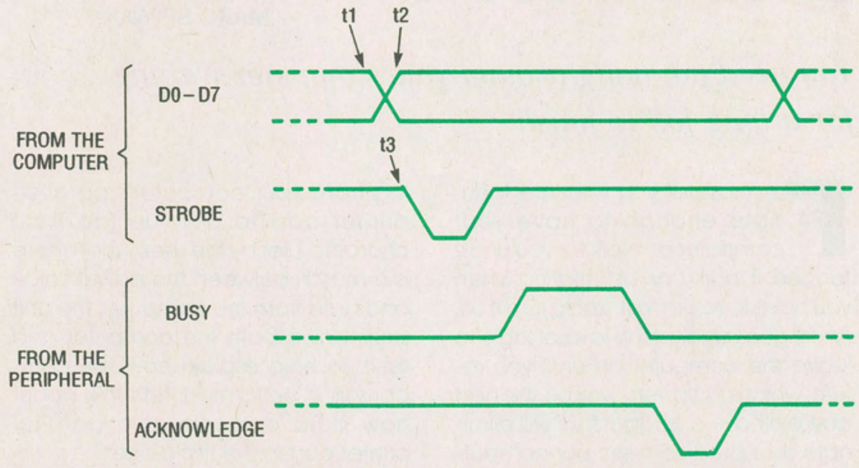


Fig. 2. Once a computer sets up the data on the parallel data lines, it signals the peripheral with a low-going strobe pulse. The peripheral responds by sending the busy line high until it's ready for more, and/or it waits until its finished with the data and sends a low-going acknowledge pulse.

can affect the flow of data, this can also be considered a form of handshaking. For example, if a printer, plotter, or oscillograph needs to tell the computer that it's out of paper, it can do so by holding the "paper empty" line (look back at Fig. 1) high until its supply is replenished. That keeps the computer from sending data to the peripheral when the device is incapable of doing anything with it.

A peripheral can tell the computer that it's powered-up and on-line by holding the "select" line at pin 13 high. (Note that there are two select lines, so don't confuse them.) This signal is necessary because some peripherals can be powered up, but taken off line by sending them a special "deselect" character. A peripheral can even cry for help by holding the error line low.

The remaining lines (select, autofeed, and initialize) are beyond the scope of this article. But now that you know how data appears at the parallel port, let's talk about how a device might hunt for the desired data and ignore all else.

Character Detection. The theory behind detecting the proper data or character at the parallel port is pretty simple. Let's say, for instance, that we wanted to detect the ASCII character represented by the binary number 11111111 (character 255) at the printer port. An easy way to do that would be to connect pins 2–9 on the parallel port to an eight-input AND gate. The output of the AND gate will then go high only if all eight inputs are high. Now, if we were to use an eight-input NAND gate instead, the only difference would be that the gate output would go *low* when all inputs are high. Since we want the gate's output to activate another device when triggered, a NAND gate is actually preferred over an AND gate, because TTL IC's are very poor at supplying current to other devices, but they are pretty good at sinking it.

That arrangement is fine if you are only interested in detecting ASCII character 255, but what about characters with binary zeros in them? The obvious answer is to invert each bit that should be zero before sending it to the NAND gate. However, using inverters is a bad idea because you'll have to rewire the circuit whenever

TABLE 1—CHARACTERS AND BIT VALUES

Bit Values:			D7	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1
			D6	0	0	0	0	1	1	1	1	0	0	0	0	1	1	1	1
			D5	0	0	1	1	0	0	1	1	0	0	1	1	0	0	1	1
			D4	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1
D3	D2	D1	D0																
0	0	0	0	(1)	►	(2)	0	@	P	`	p	Ç	É	á	░	└	╨	α	≡
0	0	0	1	☺	◄	!	1	A	Q	a	q	ü	æ	í	▒	┴	╤	β	±
0	0	1	0	☻	↕	"	2	B	R	b	r	é	Æ	ó	▓	┬	╥	Γ	≥
0	0	1	1	♥	‼	#	3	C	S	c	s	â	ô	ú	│	├	╙	π	≤
0	1	0	0	♦	¶	$	4	D	T	d	t	ä	ö	ñ	┤	─	╘	Σ	⌠
0	1	0	1	♣	§	%	5	E	U	e	u	à	ò	Ñ	╡	┼	╒	σ	⌡
0	1	1	0	♠	▬	&	6	F	V	f	v	å	û	ª	╢	╞	╓	µ	÷
0	1	1	1	•	↨	'	7	G	W	g	w	ç	ù	º	╖	╟	╫	τ	≈
1	0	0	0	◘	↑	(	8	H	X	h	x	ê	ÿ	¿	╕	╚	╪	Φ	°
1	0	0	1	○	↓	)	9	I	Y	i	y	ë	Ö	⌐	╣	╔	┘	Θ	•
1	0	1	0	◙	→	*	:	J	Z	j	z	è	Ü	¬	║	╩	┌	Ω	·
1	0	1	1	♂	←	+	;	K	[	k	{	ï	¢	½	╗	╦	█	δ	√
1	1	0	0	♀	∟	,	<	L	\	l	¦	î	£	¼	╝	╠	▄	∞	ⁿ
1	1	0	1	♪	↔	–	=	M	]	m	}	ì	¥	¡	╜	═	▌	ø	²
1	1	1	0	♫	▲	.	>	N	^	n	~	Ä	₧	«	╛	╬	▐	∈	■
1	1	1	1	☼	▼	/	?	O	_	o	⌂	Å	ƒ	»	┐	╧	▀	∩	(3)

Notes:
(1) The null character, which is not printable.
(2) The space character, which cannot be represented.
(3) Undefined character.

you want detect a different character. Not only that, you could wind up with phase-delay problems.

A chip's phase delay is the time difference between a change at the input and the corresponding change at the output. It's like when you're playing catch with someone; there is a time difference between when you catch the ball and when you throw it. That's not to be confused with rise time; the output of the chip takes time to reach the peak voltage after it has "made up its mind." Just like when you throw a ball up in the air; it takes a certain amount of time for the ball to reach its peak altitude after it has left your hand.

So, if an inverter is used along one of the data lines, it will slow that line down so all the bits will not arrive at the NAND gate at the same time. If the phase delay is long enough in comparison to the transmission speed, the device will fail to function properly.

What we need, then, is a gate that can be used as an inverter or a buffer. That way, signals that need inverting will pass through the same number of gates as signals that are simply buffered. Further, if the gate can be "programmed" to be an inverter or a buffer with a simple switch, then there will be no need to rewire the circuit to detect a different character. Simply set the switches to invert the lines you expect to be low, and buffer the rest.

The perfect gate for our purpose is the XOR gate. If you hold one of an XOR gate's inputs high, the compliment (inverse) of the other input appears at the output. If you hold one input low, the signal at the other input will be buffered through unchanged.

There are a few chips that actually contain a combination of XOR gates and a NAND gate configured to compare bits just as we've described. We used one such chip in the Kill Switch. Denoted the 74521, that chip's internal arrangement and pinout is shown in Fig. 3. Note that the chip contains XOR gates and a NAND gate wired as described with an additional input (at pin 1) that acts as a chip-enable pin. If two eight-bit numbers, A (composed of bits A0–A7) and B (made up of bits B0–B7) are applied to the indicated inputs, and if A equals B and the G input is held low, the output (pin 19) of the chip will go low. Now let's discuss how the chip is implemented.

The Circuit. The schematic diagram for the Kill Switch is shown in Fig. 4. One obvious feature of the circuit is the straight-through cable formed by PL2 and SO2. That cable allows data from the computer (attached to PL2) to flow freely to any printer cable connected to SO2. That permits the printer to operate normally without any hindrance from the Switch. If you will not be using a printer along with the unit, you can dispense with SO2, but you might have to further modify the cable to eliminate handshaking problems (more on that later).

Note that the strobe and data lines are tapped off and sent to the 74521. The strobe line is used to enable the chip (when the strobe goes low to

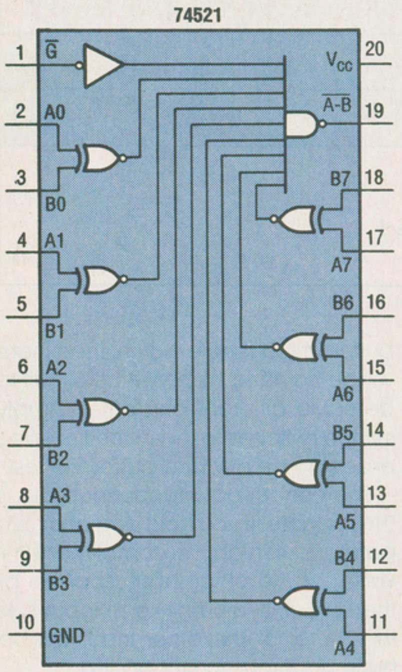


Fig. 3. The 74521 is technically known as an 8-bit identity comparator. Its output will go low provided each of its A inputs matches its corresponding B inputs and the enable input (G) is low.

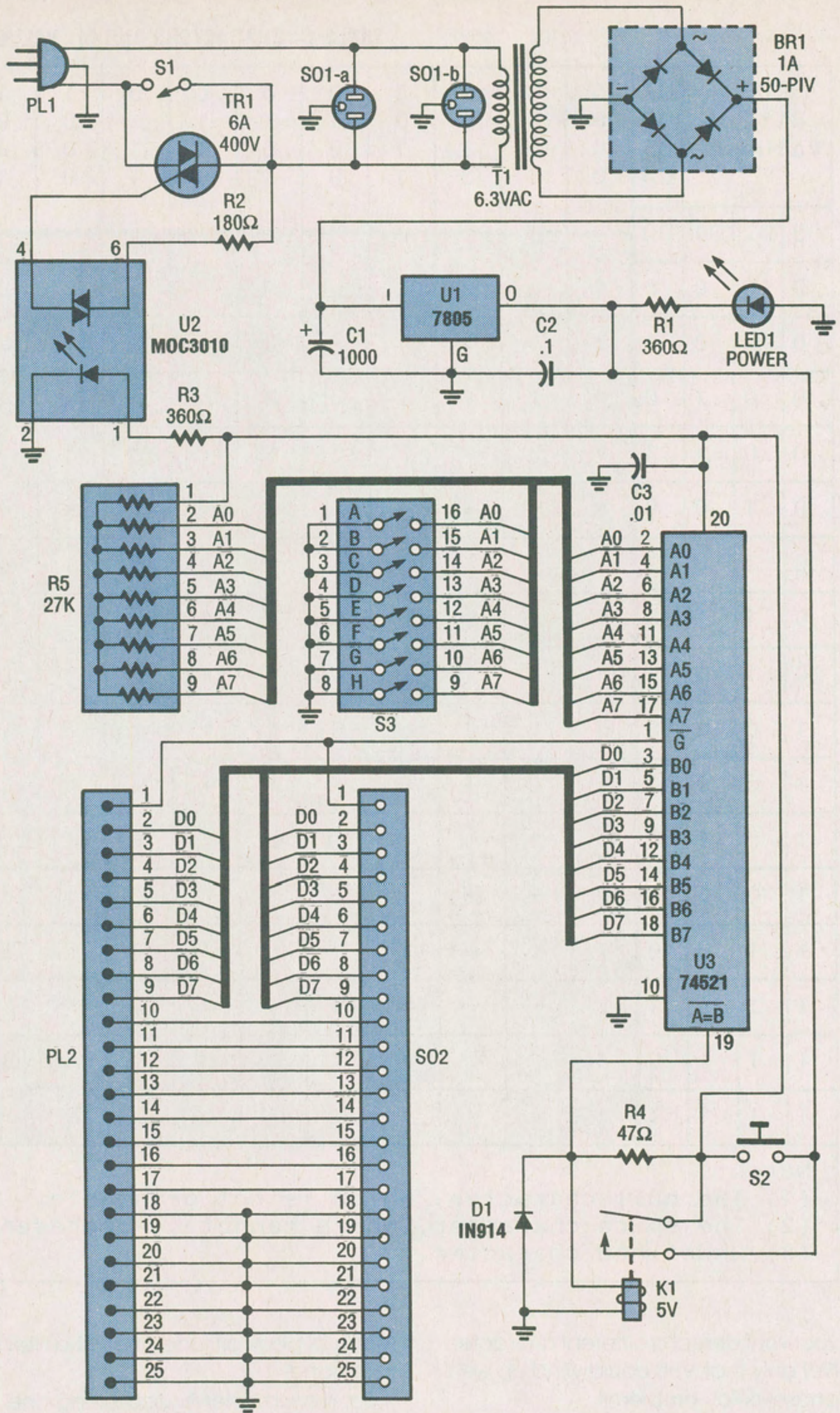


Fig. 4. At long last, here is the schematic diagram for the Kill Switch. Note how the 74521 is connected directly to the data lines.

indicate the presence of data, the chip is enabled), while the data lines feed the B inputs of the chip. The B inputs are compared with the A inputs, which are set by the ganged switches S3-a through S3-h as follows: If a switch is closed, it grounds its corresponding input, and if a switch is open, the input in question is pulled high by one of the pull-up resistors (R5-a through R5-h). In effect, each time data is present on lines D0 through D7 and the strobe is brought low, the 74521 compares the data against the switch settings. To see what affect the result of the comparison has, let's examine the rest of the circuit.

Imagine that PL1 is plugged into a wall socket and S1 is closed. That allows power to flow to sockets SO1-a and SO1-b (two halves of a duplex AC socket used to power a computer and an optional peripheral), and to the 5-volt power supply composed of T1, BR1, C1, U1, and C2. Diode LED1 acts as a power indicator for the 5-volt supply.

Note that the normally open contacts of K1 prevent current from flowing to the rest of the circuit at this point. That can be resolved by depressing S2, which momentarily applies power to the remainder of the circuit including the relay coil. The relay then closes so S2 can be released without a loss of power. With the power applied, current flows through the LED in the optocoupler activating the Triac output and the Triac.

Let's say that we turn off (open) S1, which will not affect the circuit because it is shorted by the Triac; now the system is armed. If any data from the parallel port matches the switch settings while the strobe line is low, the output of the 74521 will go low depriving the relay coil of power. The relay will immediately let go, depriving the main circuit (including the optocoupler's LED) of power. The optocoupler will then shut down so the Triac goes off. The sockets are then powerless so the computer and peripheral shut down, and after the filter capacitor (C1) discharges, the power LED goes out. That's how the unit does its job.

LISTING 1

```
MOV          DX,03BC
MOV          AL,FF
OUT          DX,AL
ADD          DX,+02
MOV          AL,01
OUT          DX,AL
JMP          0100
        A

MOV          DX,0378
MOV          AL,FF
OUT          DX,AL
ADD          DX,+02
MOV          AL,01
OUT          DX,AL
JMP          0100
        B

MOV          DX,0278
MOV          AL,FF
OUT          DX,AL
ADD          DX,+02
MOV          AL,01
OUT          DX,AL
JMP          0100
        C
```

Triggering. Since you can send data through the parallel port in a variety of ways, the Kill Switch can be tripped in just as many ways. For example, you could send the switch data from a program, DOS, keyboard, batch procedure, or word processor.

To trip the unit from a BASIC program, use a statement like:

```
LPRINT A$
```

where A$ is a string variable containing the data to trip the Switch. From any other language, any output statement that can send data to the parallel port will work.

Take a look at Listing 1 for some assembly language programming examples. Each program addresses one of the three printer ports available on a PC or compatible. Other than that they are identical, so we'll only explain how the first one works. By the way, if you don't understand how parallel ports are addressed, check out the article "Programming Parallel Printer Ports , in the February, 1993 issue of **Popular Electronics**.

The first line simply places the address of the first printer port (3BC in hexadecimal) in the DX register. The next instruction places the value 255 (FF in hexadecimal) in the AL register. That value will be used in this example to trip the unit, so all the switches in the device must be open for that value to work. (If necessary, alter that value to accommodate the switch setting you wish to use.) With the registers properly set, the third line uses the data in the registers to output value 255 to port 3BC. All that makes the data lines (D0 to D7) go high.

The next line adds two to the address to give the address that controls the strobe line (3BE). A one must be placed in that address to make the strobe line go low, so a one is next placed in the AL register. The next output instruction sends the number one to the address for the strobe line. Once the strobe line goes low, the Kill Switch should pull the plug.

There is a chance that the unit will not turn off immediately. That is because the AC waveform through the Triac must go to zero before the Triac will shut off. If the data at the port triggers the 74521 while the AC wave is not at zero, the Triac may not shut off. For that reason, the last line of the program causes it to try again and again until the computer shuts down. That generally takes only a couple of high-speed passes at most.

If you wish to use any of these programs but do not own an assembler,

(Continued on page 88)

PARTS LIST FOR THE KILL SWITCH

SEMICONDUCTORS

U1—7805, 5-volt regulator, integrated circuit
U2—MOC3010 optocoupler, integrated circuit
U3—74521 8-bit identity-comparator, integrated circuit
TR1—6-amp 400-volt Triac
D1—1N4001 1-amp, 50-PIV, rectifier diode
BR1—1-amp, 50-PIV, fullwave bridge rectifier
LED1—Miniature red light-emitting diode

RESISTORS

(All fixed resistors are ¼-watt, 5% units.)
R1, R3—360-ohm
R2—180-ohm
R4—47-ohm
R5—27,000-ohm, 8-resistor SIP pack

CAPACITORS

C1—1000-µF electrolytic capacitor
C2—0.1-µF monolithic capacitor
C3—0.01-µF monolithic capacitor

ADDITIONAL PARTS AND MATERIALS

K1—Miniature 5-volt, SPST relay
PL1—AC plug and linecord
PL2—DB-25 male connector
S1—SPST switch
S2—SPST, normally open, momentary-contact switch
S3—8-station DIP switch
SO1—Duplex AC socket
SO2—DB-25 female connector
T1—6.3-volt, 300-mA power transformer
Perfboard materials, cabinet, 25-conductor ribbon cable, wire-wrap sockets, solder posts, wire-wrap wire, bus wire, solder, etc.

Note: Copies of the software in Listing 1 are available already assembled on floppy disk for $5 (postage paid) from John Yacono, P.O. Box 4042, Farmingdale, NY 11735. Specify floppy size (5¼ or 3½ inches) when ordering. All payments in U.S. funds only; NY residents must add appropriate sales tax.

COMPUTER KILL SWITCH

(Continued from page 45)

you can use the DEBUG.COM program supplied with DOS to directly assembly the programs. See your DOS manual for instructions on using the DEBUG.COM program. Alternatively, you can order a copy of the programs compiled on diskette from the supplier mentioned in the Parts List.

The programs can simply be run from DOS, which sounds a bit inadequate, but let's examine what you can do with these simple programs and DOS. Let's say your word processor is printing a long document in your office and you'd prefer beating rush-hour traffic rather than waiting for the printout to finish. While your word processor is busy with the print job (and ignoring the keyboard), just type in the command to exit to DOS and enter the name of whichever program you will use from Listing 1 followed by a carriage return. Arm the Switch by pressing S2, turn off S1, get your coat, and run for the parking lot. When the word processor is through printing it will process the keystrokes that allow it to exit to DOS, DOS will process the keyboard input to run the program, and the computer (and the printer if it's plugged into the unit) will shut down for the night.

You can use that technique when dealing with almost any prolonged software process—faxes, modem communications, CAD computations, etc. We use it to shut our system down at night after some CAD software routes a large PC board using multiple routing strategies. Large double-sided boards can take a long time, and since we ask the computer to try it twelve different ways (typically starting late at night), the process sometimes ends in the wee hours of the morning. We like our work, but not enough to sit in front of a computer all bleary eyed just to shut it off after four or five hours.

There are yet other ways to get DOS to control the Kill Switch. For example, you could use DOS's print command to transmit a file containing the trigger character. You could also use DOS to run a batch file containing an appropriate command or two to send the data to the port. You could even use DOS to give you keyboard control over the interface. To do that type:

```
COPY CON: PRN:
```

at the DOS prompt. That command will cause anything that you type at the keyboard to pass to the printer port. You can now type-in the trigger character and send it to the printer by pressing the control and Z keys simultaneously followed by enter.

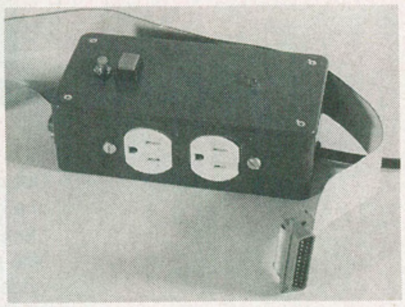
You can use a word processor, too. You can place the trigger character in a file and print the file from the word processor. That is a particularly good technique if you want the computer to shut down after printing out a file; just place the trigger character at the end of the file.

Construction. The authors' prototype was built into a relatively small case considering everything that's packed into it. All of the small components are mounted on a piece of perfboard and point-to-point wired together.

To make the straight-through connection between the computer and printer, we attached PL2 (the male DB-25) to the free end of a 3-foot ribbon cable, and attached SO2 (the female DB-25), mounted on one end of the case, about 6 inches short of the other end of the cable. The extra 6 inches of ribbon cable makes the connections from the computer/printer combination to the rest of the circuitry. The LED and switches S1 and S2 are mounted on the top of the case.

The linecord from PL1 enters the case at the other end, and the ground and neutral leads connect directly to the AC sockets (SO1-a and SO1-b). The hot lead from PL1 connects directly to S1 and TR1, which are mounted on the perfboard. The output of TR1 is then connected directly to the hot side of SO1-a and SO1-b. The openings for the AC receptacles were cut in the case using a nibbling tool. After drilling a small hole, the nibbling tool allows you to carefully cut an opening in almost any project case. No electronics hobbyist should be without one. Radio Shack sells a pretty good nibbling tool for about ten dollars.

As mentioned earlier, you can use the device without a printer. That should cause no trouble for those of you who will control the switch using


Here's the authors' completed prototype for their computer-controlled Kill-Switch.

the software in Listing 1. However, if you will be using software that relies on DOS and/or the computer's BIOS, you could run into trouble unless you modify the Kill Switch. Unlike our program, DOS and the BIOS might look at the various handshaking lines to make sure the "printer" is okay. Of course, since there is no printer, DOS and the BIOS will assume something is wrong and never send data to the port.

One way around that is to trick the computer into thinking that there is a ready and waiting printer available by tying the handshaking inputs to appropriate logic levels. Specifically, tie pin 12 (paper empty) and pin 11 (busy) low, tie pin 13 (select) and pin 15 (error) high, and connected pin 10 (acknowledge) to the strobe line. Note: the strobe line should still also be connected to the 74521.

Now you might be wondering why you should tie the acknowledge line to the strobe line. That permits the computer to "shake its own hand." When the computer sends the strobe pulse, it considers the falling edge the start of transmission (that's indicated by the arrow on that edge). By the time the strobe line is low, the computer has already begun waiting for the acknowledge pulse. Interestingly enough, the computer only concerns itself with the rising edge of the acknowledge pulse; it pays no attention to the pulse's logic level or its falling edge. Since the computer starts waiting for the rising edge of the acknowledge pulse right after the falling edge of the strobe pulse, we can use the rising edge of the strobe pulse in place of the rising edge of the acknowledge pulse. That's what the strange connection accomplishes.

Once you have a working Kill Switch, you'll wonder how you ever got along without one. ■