14278

INS-AND-OUTS OF COMPUTERS FOR BEGINNERS

Popular Electronics®

WORLD'S LARGEST-SELLING ELECTRONICS MAGAZINE

JUNE 1976/$1

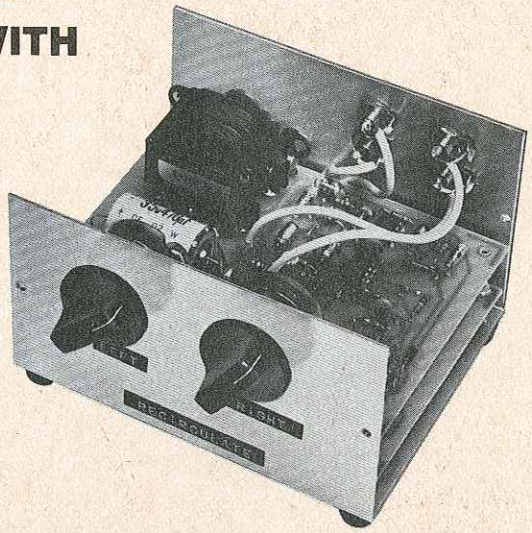
AUDIO * STEREO * MUSIC

ADJUST ROOM REVERBERATION WITH

The "Bucket Brigade" Audio Delayer

PE TESTED

A $59 STEREO PROJECT



Ten Speaker-Enclosure Fallacies Exposed!

"Music Modules" Simplify Synthesizer Kit

PROJECTS FOR SUMMER USE

Now You Can Operate AC Equipment from a Battery!

SINE-WAVE INVERTER BOOSTS 12 VDC TO 110 VAC

Build a Low-Cost ilure Alarm

Test Reports:

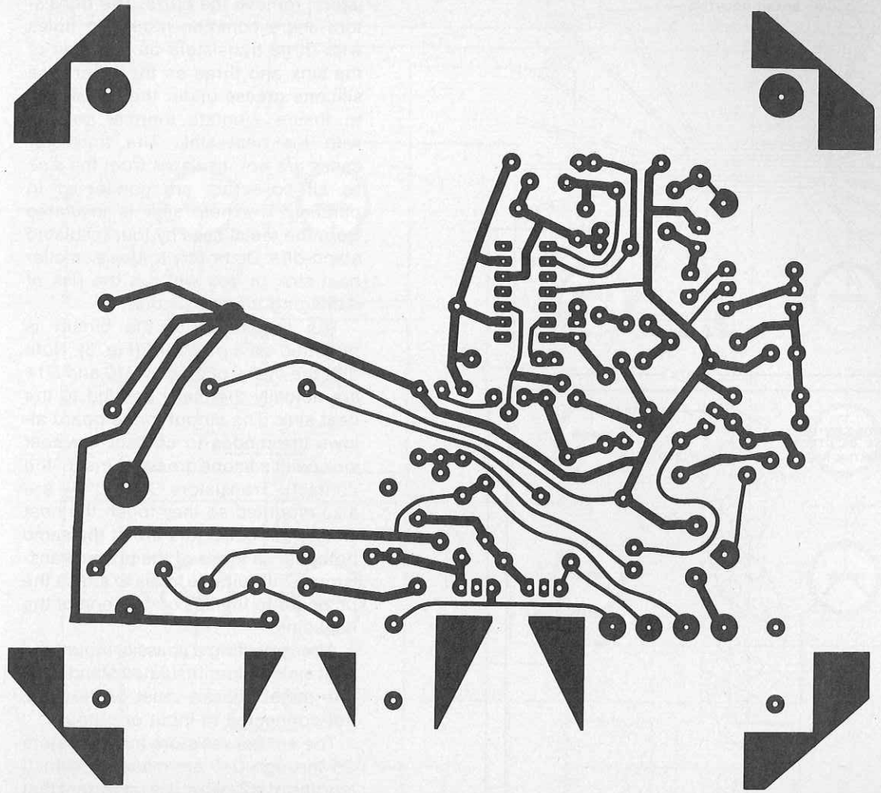
HEATH MODULUS AM/STEREO FM SYSTEM

SOUND CONCEPTS SD-50 AUDIO DELAY SYSTEM

CRAIG 4104 CB AM MOBILE

CONTINENTAL SPECIALTIES DESIGN MATES

ZIFF-DAVIS PUBLISHING COMPANY



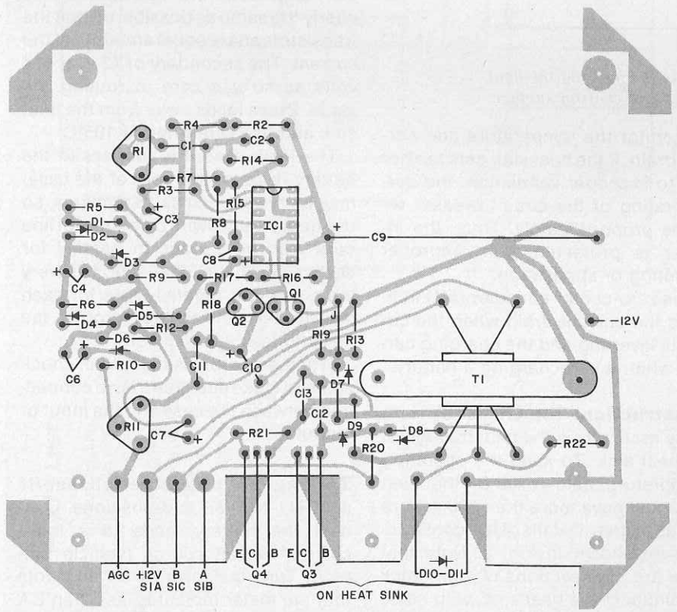


Fig. 3. Etching and drilling guide and component layout for pc board. D10, D11, Q3, and Q4 touch heat sink.

If the meter indication is correct, turn off the inverter and connect a 117-volt ac meter and a 100-watt lamp to *SO1*. Keep in mind that this is a hazardous voltage. Turn the inverter on and adjust *R11* to obtain 117 volts at *SO1*.

Use a frequency counter or the circuit shown in Fig. 4 to adjust *R1* for 60 Hz. In using the circuit in Fig. 4, adjust *R1* until the neon lamp does not flash (zero beat).

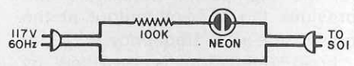


Fig. 4. Use this circuit to tune the inverter to 60 Hz.

Operation. This equipment, like any ac line-powered gear, must be treated with great care. The cabinet should be adequately ventilated at all times. The design is safe up to an ambient of 120°F. If the circuit breaker trips, check the ventilation and possibly reduce the output voltage slightly. It is good practice not to operate any electronic gear in an ambient in which a human is not comfortable. ◈

IN's and OUT's of COMPUTERS for BEGINNERS

BY EUGENE H. MITCHELL

Understanding character codes, flags, interrupts, DMA, and other computer terms.

SO YOU HAVE finally found what you hope is the last solder bridge on your homebuilt computer, put the case on, and turned the dining table back to your wife. Now you are ready to start using your computer; but after one long evening of working the switches and watching the lights, you realize you don't really know how.

Did you read the operating manual? If so, you would have found that there are a number of "input/output ports" available. However, you can't just feed data through an input port and expect it to come flowing from the output port. You have to have some peripheral devices to attach to those input and output ports.

A peripheral device can be a teletypewriter, card reader, paper-tape punch, CRT terminal, etc. Magnetic tape and discs are also part of the peripheral device scene. However, these devices don't just sit there and communicate with the computer automatically. You have to know how they work and how to "talk" to them through the input/output ports.

Every device has its own idiosyncracies. There are two main characteristics that we will consider here: character codes and speed.

Character Codes. The easiest way to get a good feel for the concept of a character code is to design an input device. First, we must decide upon its alphabet; that is, we must describe *precisely* the entire set of characters that this device will recognize. We then order these characters in whatever arrangement suits our fancy and number them from 0 to *n*, where *n* is the number of distinct characters in our alphabet.

The device is now constructed so that when it recognizes a specific character, say the 17th character in its alphabet, it transmits its *number*, 17, (in binary, of course) to the computer's input port.

The question arises: how many bits are needed to uniquely code an alphabet? The answer is that we need at

least $log_2 n$. Conversely, if a character code contains n bits per character, then the maximum number of characters is 2^n. Thus, an 8-bit code can describe a 256-character alphabet.

Another question is: how many characters do we need in an alphabet? In English, we need 26 letters, 10 digits, a number of punctuation characters, and a *blank*. Never forget that a *blank* is a character! If we allow for eleven punctuation characters we find we need a total of 48 characters. Note that we have 26 letters with no discrimination between upper and lower case. If we want both cases, we must add another 26 characters—upper and lower case of a given English letter are two completely different characters to a computer! The total is now 74 characters.

Five bits would give us 32 characters which is not enough. Six bits would permit 64 characters, so 6 bits is the minimum number we need for a reasonable alphabet, although we need at least 7 if we are to recognize both upper and lower case letters. For years the 6-bit code was a default industry standard and the default character set was the 48 characters available on the IBM model 026 keypunch. When IBM introduced the 360 computer they went to a model 029 keypunch with 64 characters. The computer, however, used an 8-bit character code.

You may have heard of a character code known as ASCII (American Standard Computer Information Interchange) which is used in the newer teletypewriters. This has an 8-bit code and 128 characters including upper and lower case. However, most teletypewriters have only upper case letters.

To build an output device, we go through a similar procedure. The major difference is that when the computer gives it a binary number, the output device produces the corresponding character of *its* alphabet, not necessarily the same as that of the input device.

Data Rate. Let's say the output device is a typewriter with a speed of 10 characters per second. Let's type the letters "AB". First, we put the code for an "A" in the output port, followed by the code for a "B". In your computer this would take a few microseconds. But it takes the device 1/10 of a second to type the A! Thus, it would do one of two things: type the A and never see the B, or never see the A and type out the B.

What we must do is put the code for the A in the output port, then do something else for 1/10 of a second (we may just have to waste 1/10 of a second by looping), then put the code for B in the port.

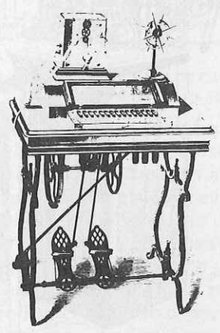
Let's leave the computer and look at the output port from the eyes of the device. First it "sees" the code for an A so it prints an A; then it looks back at the port and sees the code for a B and prints a B; then it looks back at the port and sees the code for a B (it's still there!) so it prints it. And so on, ad nauseum. We obviously need some method to avoid this. One way is to define a character code which means "do nothing" (this is NOT a blank). We will give this code a name; it is the *null* character. Then to send "AB" we output A (we really mean we output the code for an A), wait, output B, wait, then output *null*. The device prints the A, prints the B, then continuously does nothing so long as the *null* remains in the port. But this process can be improved.

One way to solve our problem is to use a special flip-flop, called a flag, for each output port. As long as the flag is reset (zero), the output device does nothing. If the flag is set, the device outputs the character from the output port and *resets the flag*. Thus, in general, we no longer need a *null* character; the flag bit takes its place. The control pulse generated by the computer to load the data into the output port is also used to set the flag.

We gained something else. If the computer can somehow determine the state of the flag bit, it can tell if the character previously in the output has yet been accepted by the device. This means that we no longer have to waste a programmed amount of time and *assume* the device has processed the character, but we can *verify*, by testing the flag. This is important when different models of the same computer run at different speeds. An Intel 8008-1 running at full speed cannot properly drive a teletypewriter using a program developed on a standard 8008, because the do-nothing loop has been programmed to waste the correct number of cycles for the 8008 clock. The do-nothing loop which tested the flag would work on either computer, because the device supplies the timing. This means that each output port must be associated with an input port to input the flag.

We must arrange the power-up logic so that all the flags are initially reset to prevent the output devices from outputting garbage when the machine is first turned on. Also, it would be nice to have a master clear button to force all the flags to zero if we need to manually stop and restart the computer. In some cases, it would be useful to have a special output port connected to the same master clear logic so the program could reset all flags with one instruction. We may also want the ability to set or reset individual flags under program control. This would require another output port for each output device. Thus, a full-blown flag facility would require two output ports and one input port for each output device. Similarly, it would need two input ports and one output port for each input device. Actually two output ports and one input port could handle all the flags if you dedicate one input and one output port to reading a flag and writing (setting or resetting) a flag, and another output port to specifying which flag is to be read or written.

Now let's look at an input using the flag bit. The input device would read a character, place its code in the input port, and set the flag. The program would then test the flag, find it set, and then read the character from the input port, resetting the flag. The problem here is that the operation is initiated by the device, not the program (this may be desirable in some applications). Usually, we don't want the input device to function until the program invites it to. This implies that we need two bits—one to tell the device to operate, and one to tell the computer that the operation is complete. Let us use our flag bit for the latter function and add an additional bit (a control bit) to control the device.



Input keyboard?

Flag bits.

For output:

1. Set the control bit to start the device.
2. Put data in the output port, and reset the flag.
3. The device accepts the data and sets the flag.
4. The computer repeats from step 2 until all the data has been output.
5. The computer resets the control bit to stop the device.

For input:

1. Set the control bit to start the device and reset the flag bit.
2. The device puts data into the input port and sets the flag.
3. The computer reads the data from the port and resets the flag.
4. This repeats from step 2 until all the data has been input.
5. The computer resets the control bit to stop the device.

Call Me, I Won't Call You. If you are working when the phone rings, and you stop to answer it, you have been *interrupted*. The caller may have an urgent request for some information in which case you suspend what you are doing to supply the information, then return to your original task. On the other hand, the call may be simply to inform you that you are to be in a meeting at a later time. In this case, you post the request on your memo pad or calendar and at the proper time, stop what you are doing and attend the meeting. The call may also be to inform you that something you had previously requested of the caller has been completed. Such a facility, called *interrupt*, can be built into a computer.

If you have only one phone on your desk, you must ask the caller to identify himself when the phone rings, since many different people can interrupt you over the same phone. In a computer, this method is called a *basic interrupt* facility. Another method is a *vectored interrupt* facility. To illustrate, you may have a desk full of phones, with each number known to only one prospective caller. In this case, it is not necessary to have the caller identify himself. You know who it is by which phone is ringing. Alternatively, a small number of people may be given the same number so a particular phone does not uniquely identify the caller, but limits the possible callers to a small set. To implement an interrupt facility within the computer, we do the following:

1. Save the address of the next instruction in some specific place; in machines with a stack, it is usually convenient to stack it.
2. Force the program counter to a specific address.

In a *basic interrupt* facility, the address used in step 2 is the same, regardless of who interrupted. In a *vectored interrupt* facility, the address used is a function of the particular interrupt. (See the restart instruction on Intel machines.)

Now we can produce an even better input/output system than we have to date. Just wire the flag bit *true* output to the interrupt line. Actually, we should probably AND the flag and control bits and wire this to the interrupt line. Then, when the control bit is set (the device has been invited to operate) and the flag bit is set (the device has operated) the computer will be interrupted. Thus, we no longer need a timing loop to assume the operation is complete nor a test loop to verify the operation is complete. Simply go on with the program or go into a do-nothing loop until the interrupt occurs. No test is needed since the occurrence of the interrupt will automatically tap the computer on its shoulder and give it the address of another program to execute. This other program is called the interrupt service routine.

A Mind-Reading Machine. You may have noticed that we have been sending only one character at a time. Usually, a peripheral device has a line length that it prefers. Punched cards generally use 80 characters, printers 132, typewriters 72 to 90, etc. In many cases, these are specified in terms of the maximum number of characters. Such a group of characters is called a *record*. Some devices must transmit an entire record at a time, while others transmit *incrementally;* one character at a time. There is usually some character in a hard-copy machine's 'alphabet' which does not print, but causes the carriage to be returned, thus defining the end of the record. Some devices, such as discs, are built so that, once a transmission is started, an entire record must be transmitted. For this reason, a common method of programming prepares the entire record in the computer's memory, and then sends one character at a time. If the entire record is in memory, there must be a better way to output it!

Suppose we built an "intelligent" output port that operated as follows: In lieu of putting the individual characters in the port, we give the port the address of the first character and tell it the number of characters to transmit. Then we allow the "intelligent" port to reach into memory at its own speed to fetch the characters and pass them on to the device. Such an intelligent port is called a *direct memory access* device, since it reaches directly into memory as it performs its function.

But if the DMA is accessing memory and the computer is accessing memory, can't things get fouled up when they both attempt to access at the same time? They sure can! To use such a facility, the computer must be appropriately designed. Read your manual again. If you have a machine built around the Intel 8080 you will find a pin called HLDA (hold acknowlege) and another labeled HOLD. When HOLD is raised, the computer finishes its current cycle, switches the address and data buses to the high impedance state, and raises the HLDA line. At this point the memory bus is available to the DMA with no interference from the CPU (central processing unit). When HOLD is dropped, the CPU resumes its execution. Thus, the DMA can directly access the memory for either input or output. If the device attached to the DMA is medium- to low-speed, the HOLD line is dropped after every access to permit the CPU to operate. If the device is fast, the DMA can lock the HOLD line high and seize the memory for the duration of the transfer.

There are a number of methods to implement this DMA. The simplest assigns a particular address to each DMA when it is built or installed. The programmer must put the first character of each record for the DMA device at this address. Then, all he has to do is set the control bit and reset the flag, through an output port, to start the

transfer. When the operation is finished, the DMA sets the flag, and, optionally, an interrupt occurs. The problem here is that each record for the device must start in the same location and must be the same length.

A more flexible arrangement uses an output port to feed the address and the record length to the DMA. This would require a transmission of 4 characters in the proper order. At this time the DMA could start its function. Again, when the operation was finished, the flag would be set and/or an interrupt could be requested. An even more flexible arrangement would permit several devices to be attached to the DMA. The program would output 5 characters to the DMA to start it (2 characters for the address, 2 for the number of characters to transfer, and one to identify the device and specify the direction, input or output). Using an 8-bit character, we could use 1 bit to specify the direction (0 means output and 1 input, say) and the other 7 bits to specify one of 128 devices. By using an additional bit to inform the DMA whether to interrupt or not gives us more flexibility and still permits us to handle 64 devices. Of course, only one device can be in use at any given instant. A DMA such as that described above is sometimes referred to as a basic input/output channel. Note that, while the DMA is functioning, the CPU can still operate, possibly at a reduced speed due to the fact that the DMA is stealing memory cycles from the CPU. If the DMA is locking the HOLD line high, the CPU cannot operate, of course, until the transfer is complete.

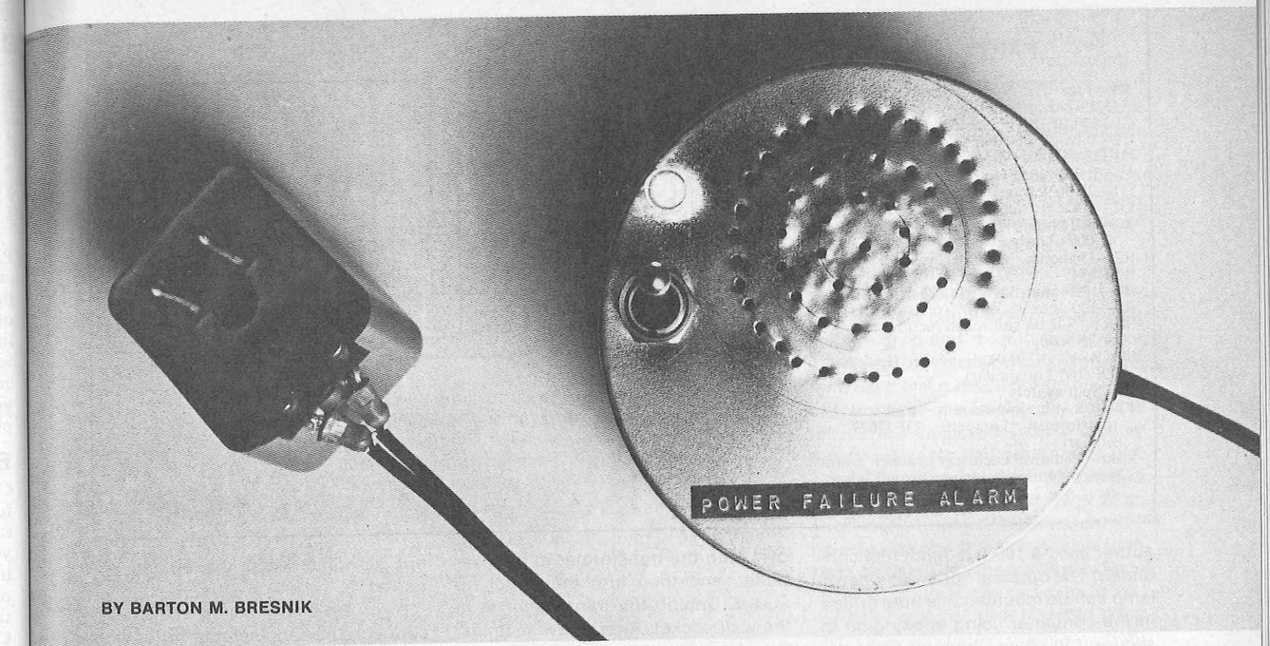
Since the DMA can effectively lock out the CPU, we must arrange the power-up and master clear logic so that no DMA will attempt operation until the CPU directs it to.

Control And Status Signals. There are two types of transmissions yet to be considered: *control* signals sent to the device, and *status* signals received from the device.

Control signals cause the device to perform non-data operations such as start a new line, start a new page, backspace, etc. There are two ways to send such signals to the device. The first is to have a separate control path to the device from the computer, while the second is to define certain alphabetic characters as control characters rather than data to be printed. It is this second method that is used in ASCII. In many cases, the same bus which carries the data also carries the control signals, but a secondary line is raised to indicate that the information is control data and not alphabetic data. Control signals are used to cause tape drives to rewind or backspace over a record, a disc drive to select a different track, a hard-copy machine to return the carriage (and, usually, start a new line), printers to skip to a pre-defined spot on the page, erase the screen on a CRT, etc.

Status codes are sent from the device to the computer and generally are used to convey information about the device's condition. Common items of information which are conveyed to the computer are the status of the device's power supply (on or off); an indication that it is busy (for instance, a carriage return takes much longer than typing one character); that one or more characters were sent (or received) erroneously during the last transmission; that a tape drive is at the beginning of the tape and should not be backspaced further; that a tape drive is at the end of the tape and no further attempt should be made to read or write it; etc. Status information can be treated in much the same way as control information. It can be returned to the computer over a separate bus; it can come over the data bus accompanied by a signal which identifies it as status rather than data; or it can be built into the device's alphabet.

Notice that input devices now, in general, have an output type of characteristic, so we can send an input-device control signal. In the same way, output devices generally provide for input of their status. The two can also interact. A particular device may send status information to the computer only after it has been invited to do so by a command signal. In many cases, the system protocol requires a device to send status information at the end of every transmission. In some cases, the "standard" status is assumed to be that every thing is fine unless something goes wrong. In such cases, we frequently find that the status information is wired to an interrupt so that, as long as the transmission is proceeding according to plan, nothing happens; but an interrupt occurs if an unusual condition arises. At this time the interrupt program can request the status to obtain the details of what happened. ◈



BY BARTON M. BRESNIK

POWER-FAILURE ALARM

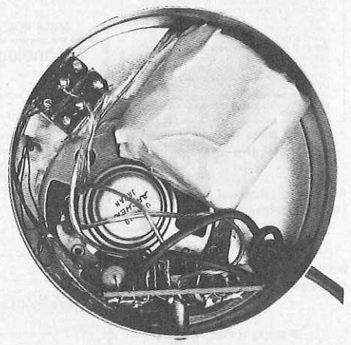
Lets you know when a power outage occurs.

SUMMER or winter, night or day, a power outage in your local utility system can cause all sorts of problems in your home. Heating and cooling systems shut down, refrigerators and freezers come to a halt, and your electric alarm clock stops, making you late for work.

The power-failure alarm is a battery-powered device that sounds an alarm when a power failure occurs. Then you can, at least, turn off devices that might blow fuses when the power returns and take what other steps are necessary to protect your property.

How It Works. Battery *B1* (Fig. 1) gets a constant trickle charge from the transformer through *D1* and *R1*. As shown here, the battery is made up of two 1.25-V NiCd cells. Sealed NiCd or lead-acid storage cells with higher voltage ratings could be used. Vented secondary batteries can be used if the electrolyte is checked every few months. If carbon-zinc or manganese-alkaline cells are used, the value of *R1* should be increased to 47,000 ohms. Remember also that manganese-alkaline and mercury cells may burst when recharged.

The alarm generator consists of a two-transistor astable multivibrator and associated loudspeaker, while the trigger portion uses an SCR and related bias components. The SCR is in a feedback loop from the emitter of *Q2*. The gate of *SCR1* is biased low enough to keep it from firing by the combination of *R3* and *R4*. When a power outage occurs, the voltage from the battery turns on the SCR, and the multivibrator provides an audio-frequency signal to the speaker.



Author's prototype was assembled in a 35-mm film container.

The time delay provided by *C1* and *R3* is used to keep the system from operating in case there is only a brief loss of power (which can be caused by lightning) or a line transient.

In standby operation, the circuit draws less than 1 mA, which is supplied by the trickle charging current. When an outage occurs, and the SCR turns on, the current increases to 15 mA for a 2.5-V battery and 50 mA for a 4.5-V source.

The lamp circuit is optional and can be used to check the battery. The lamp can also be made to glow during a power outage by connecting a silicon diode between the LAMP position of *S1* (anode of the diode) and the anode of *SCR1* (cathode of the diode).

Construction. The prototype of the alarm was assembled on a small piece of perforated board with point-to-point wiring. For transformer *T1*, use a standard recharging unit which plugs directly into a wall socket. This provides a safety feature in that only 6.3 volts is used in the chassis.

Mount the completed assembly in any type of enclosure with only *S1* and some speaker holes on the top. (The