Get Your Pet on The IEEE 488 Bus

This 3-part odyssey takes you along route 488. The first stop is here . . . tickets, please.

Gregory Yob
Box 354
Palo Alto, CA 94301

Perhaps the most obscure Commodore PET feature is its IEEE 488 (or HPIB or GPIB) interface. This three-part article describes the rudiments of the 488 bus and how to use your PET to communicate with instruments having the 488 interface. Several working examples with Hewlett-Packard equipment are shown. (HP lent me several 488-compatible instruments to prepare this article.)

If you just want your PET to talk to that costly instrument on your bench, skip this month's installment and start next time with part 2. The first two parts of this article will sketch the prerequisites and give you enough information to track down bugs on your own.

What's a 488 Bus?

In 1972, engineers – some with Hewlett-Packard – proposed a method of joining many instruments in a standardized way to help automate lab and test measurements. This resulted in the IEEE Standard 488-1975, which describes how to connect as many as 15 instruments on the same cable.

HP and several other laboratory-instrument manufacturers then offered the IEEE 488 scheme as an option. Presently, several hundred instruments have the 488 capability; Commodore used to provide a 5-page list of these. The PET was later designed with the instrumentation and control market in mind, so the IEEE 488 interface was put into the PET.

Before the introduction of the PET, instruments capable of controlling the 488 bus cost several thousand dollars. Now the PET often costs less than the instruments it controls. Some 488 manufacturers have trouble adjusting to this – their customers balk at the idea of purchasing an $800 microcomputer to control a $30,000 instrument!

Now one connector joins the PET to many peripherals. You don't need a separate interface and connector for each new gadget. Commodore's printer and disk are designed to use the PET's 488 interface.

Physical Aspects

A PET and a 488-compatible device have different connectors. Your first project is to wire a cable to tie the two machines together.

Fig. 1 shows the location of the IEEE 488 connector on the back of the PET, and Fig. 2 describes the pins and connectors used for the PET and the IEEE 488. I used a 20-conductor ribbon cable and tied the

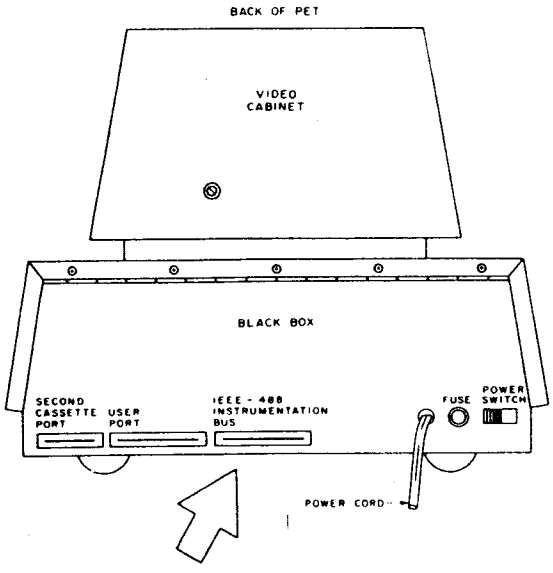

Fig. 1. Location of PET IEEE 488 port on the back of the PET next to the power switch and fuse.

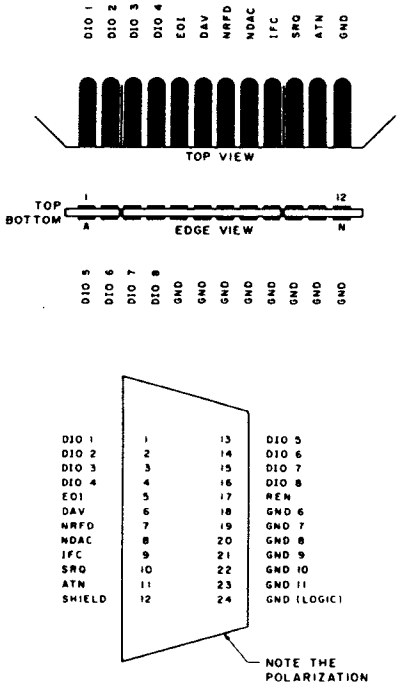

Fig. 2. Pin-outs and connectors for the IEEE 488.

grounds together into the four lines left over after I connected the signal wires.

When making the cable, bear in mind that there are strict limits to cable lengths:

1. The maximum distance between two devices is 5 meters.
2. The longest distance from one end of your setup to the other is 20 meters.
3. A maximum of 15 devices, including the PET, can be hooked together.

It is also wise to avoid electrically noisy areas; don't drape your IEEE 488 cable over your TV set.

If more than one device is connected to the 488, you must use extension cables. HP has cables for about $50. If you want to make your own, consult the two configurations in Fig. 3. The 488 instruments always have a female connector, so have an excess of male connectors on your cables.

Electrically, the 488 bus works on an active-low principle. Fig. 4 shows a circuit similar to a 488 bus line. When all the switches are open, the voltmeter will show 5 volts, which is the false state (or 0) for the line. If any of the switches are closed, the line is grounded, and the voltmeter shows zero volts, or the true state.

This peculiar arrangement permits several devices to be connected to the same line. If any one of them has a switch closed, the line is true. Devices frequently operate at different speeds, and when each device is ready, it opens its switch. However, the line remains true (low) until the slowest device opens its switch.

IEEE Blinkin Lites Display

It is always convenient to have a display and switches to perform a front panel function when you debug interfaces. I built a box, which I call the 488 Blinkin Lites, to display the states of each of the IEEE 488 lines and some switches to force lines low if needed. Fig. 5 shows the circuit, and Fig. 6 is a sketch of my box.

Each line is pulled up to +5 volts with a 10k resistor—the high value was chosen to minimize the load on the 488 bus. The switches can override any line when they are closed to ground. Though the PET doesn't use all the IEEE 488 lines, future machines will—so I put them all in my box.

If you build this box, don't use the PET's +5 volts from the tape port—the LEDs draw 170 mA, which is too much for the PET. Provide a connector to the PET's IEEE port and a male and female IEEE connector. This lets you interpose the IEEE Blinkin Lites between the PET and an instrument.

I mounted a 5×7 inch perfboard with 0.10 inch holes into a standard breadboard box and placed a label near each switch/LED combination to identify the IEEE lines. The three ICs are the 7404s used to drive the LEDs. The cable leads to a homemade junction with a PET connector and IEEE male and female connectors. A mini phono jack connects to a separate +5 volt supply (see Fig. 6).

When you plug in the IEEE Blinkin Lites, the LEDs will show the state of the lines—an LED that is off indicates a low line, which is true; an on LED indicates high, which is false.

The IEEE 488 Lines

The IEEE 488 is composed of 16 lines. Eight are for transfer of data, five are for bus management and three are for handshaking. The eight data lines are labeled DIO1 through DIO8, with the most significant bit (MSB) on DIO8. The 488 bus can transfer one byte at a time and is sometimes called byte-parallel.

The five bus-management lines in various combinations and sequences provide many bus facilities, most of which are rarely used:

EOI—End of Message. When a group of bytes is sent via the DIO lines, EOI is made true on the last byte to indicate that the message is completed. This is optional, and many instruments send the ASCII characters CR and LF as data instead. Check your instrument's manual.

IFC—Interface Clear. When this line is true, all instruments disconnect to a defined state. (This usually is unaddressed and untalked.) When you turn on the PET, IFC is true for about 100 ms. If the PET is reset, IFC will again be true.

SRQ—Service Request. This permits an instrument to signal that it needs attention . . . and the device in charge of the bus must find out what it needs. The PET has this line as an input, but it takes some programming effort to use SRQ; most instruments don't use SRQ.

REN—Remote Enable. Most IEEE instruments have front panels that permit stand-alone operation—that is, they work as ordinary instruments when the 488 bus isn't connected. REN lets the instrument disconnect from the bus and be controlled from its front panel.

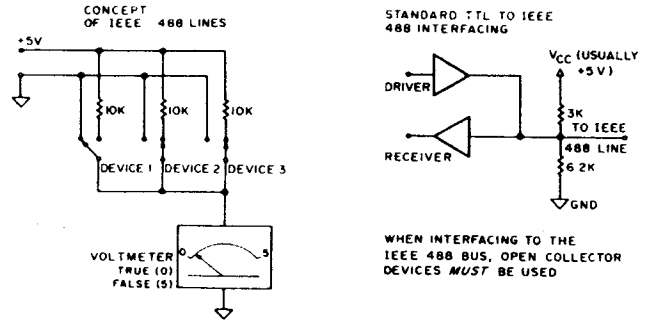

Fig. 4. IEEE 488 equivalent circuits. The lower circuit is the standard method of connecting TTL logic to the 488 bus. The driver must be an open collector and able to sink at least 48 mA at .4 volts and source 5.2 mA at 2.4 or more volts. The PET uses MC 3446P bidirectional line interface ICs for this function.

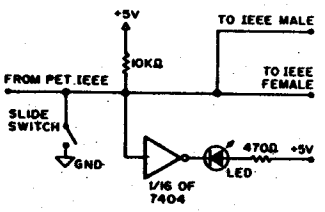

Fig. 5. IEEE "Blinkin Lites" circuit. Each IEEE line uses one copy of this circuit.

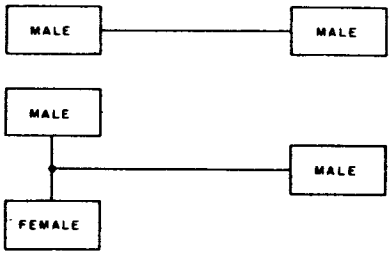

Fig. 3. Convenient cable configurations for the IEEE 488 bus.

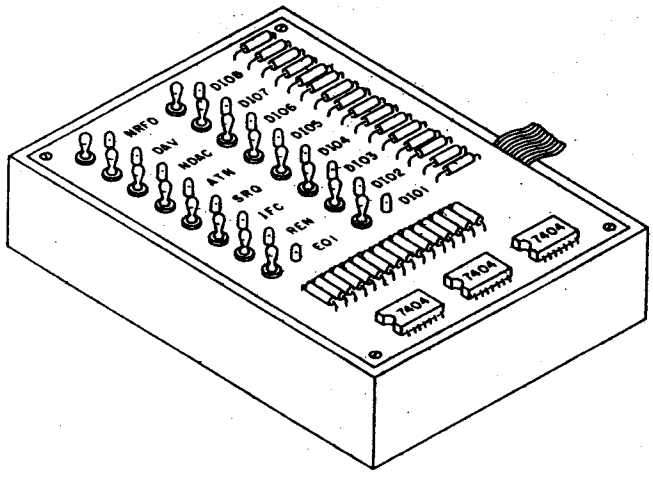

Fig. 6. Sketch of the "Blinkin Lites."

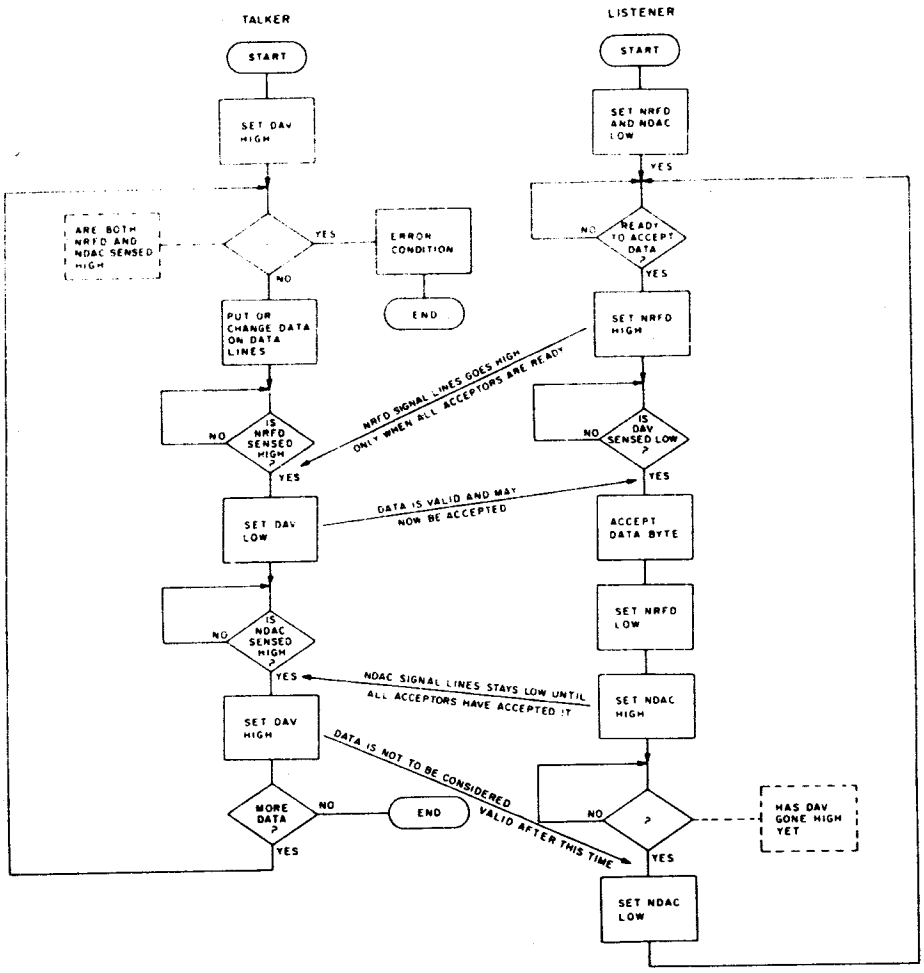

Fig. 7. The IEEE 488 handshake reproduced from Electronics, *Nov. 14, 1974, p. 98, as reproduced in HP part #5952-0058.*

The PET's REN line is always grounded.

ATN—Attention. This is the most relevant line for this article. It tells the device whether to regard the data on the DIO lines as a command or as data. When ATN is true, the byte on the DIO lines is a command. When ATN is false, DIO is seen as data.

The three handshake lines are used to pass bytes on the DIO lines. When a byte is transferred, the slow devices will keep one or more of the handshake lines true until they are finished. This ensures that data is passed at the speed of the slowest device and isn't lost. The handshake lines are:

DAV (Data Valid)—When this line is true, the data on the DIO lines is correct and the receiving instruments can pick up the byte.

NRFD (Not Ready For Data)—When a receiving device is busy or is still processing prior data, it will make NRFD true, which stops data transfers.

NDAC (Not Data Accepted)—When the data is on the DIO lines, the receiving devices keep NDAC true until all of them have read the data byte. Note that the handshake lines don't care whether the data is a command or not; every byte of data or command has to undergo the handshake sequence.

The Handshake

For data transfer, one device is the "talker," which provides the data or commands for transfer. The recipients, or "listeners," pick up the data, and more than one device may listen at the same time. The handshake specifies exactly how the data transfer is accomplished.

Fig. 7 shows a flowchart of the handshake sequence. When the first event, NRFD, goes false, this tells the talker that all of the listeners are now ready to receive a new data byte. The slowest listener is the last one to release NRFD, which will go high.

Next, the talker puts the data byte on the DIO lines and waits briefly to let the signals settle (usually about 10 μs). Once the data is on the DIO lines, NRFD is checked by the talker; if it is false, the talker sets DAV to true. The listeners now know that the new data is ready for pickup. (If NRFD is true, the talker waits until it goes false.)

The first listener that detects DAV true now sets NRFD true, and all of the listeners pick up the data byte from the DIO lines. Up to now, NDAC has been true, and as each listener gets its byte, it releases NDAC. NDAC goes false when all the listeners have the data. The talker waits for NDAC to go false, and when it does, the talker sets DAV to false. The listeners then make NDAC true, and the entire handshake sequence begins again.

Since a device is either a listener, talker or not addressed, Fig. 7 is broken into two flowcharts: one for the talker and one for the listener. A listener will start the handshake with NRFD and NDAC true, while the talker checks these. If both are false—the listener isn't there—an error condition exists.

Commands and Messages

When ATN is true, any data on DIO is seen as a command. Fig. 8 shows the entire ASCII set of 128 characters devoted to IEEE 488 commands.

The ASCII codes 32 through 62 (all numbers in decimal) designate the listen address for a device. Most IEEE-488-compatible devices have a five-position DIP switch next to the 488 connector set to the device's address, a number from 0 to 31. (Note: For the PET, use 4-15.) When the listen address is sent with ATN true and this address matches the device's address, the device will now be addressed to listen and will accept any data sent with ATN false.

If the device is supposed to send data, the talk address—from ASCII codes 64 through 94—will be used instead. The device (if with matching address) will now send data bytes to the bus.

If the device's address (by the switches) is number 7, the listen address value will be 32 + 7, or 39 (apostrophe). The talk address will be 64 + 7, or 71 (letter G). Notice that bits 5-7 designate talk or listen, and bits 0-5 designate the address. Address 31 is reserved for two special commands. Although you can set the switches on a device to 31, it won't operate with this setting.

One instrument must provide these talk and listen addresses. This device is the controller, and the PET is always the controller. The controller can talk and listen too, but only the controller can set ATN true.

Two of the ASCII codes, 63 and 95, serve as "universal" commands. The 63 code is known as "unlisten" and tells all addressed devices to stop listening to the bus. This is faster than trying to tell the devices one at a time to stop listening. The 95 code, "untalk," stops all data transmitters (talkers).

When a message—or a group of data bytes—is sent on the data is present. (In normal operation of the bus, the controller doesn't have to take these drastic measures.)

In some cases, a device will have a secondary address, which permits more than 31 effective addresses on the bus. For example, the Commodore printer might be set as device 4. To control internal functions, secondary addresses select the function in use. (See Commodore's "PET Communication with the Outside World," p. 19.) If a secondary address is in use, it is sent immediately after the talk or listen address, known as the primary address, with ATN true.

Table 1. All PET I/O lines.

IEEE 488 PIA (6520)		ADDRESS: $ E820 59424	
PAØ	IEEE Data In 1	PBØ	IEEE Data Out 1
PA1	" " 2	PB1	" " 2
PA2	" " 3	PB2	" " 3
PA3	" " 4	PB3	" " 4
PA4	" " 5	PB4	" " 5
PA5	" " 6	PB5	" " 6
PA6	" " 7	PB6	" " 7
PA7	" " 8	PB7	" " 8
CA1	ATN In	CB1	SRQ In
CA2	NDAC Out	CB2	DAV Out

MULTILINE INTERFACE MESSAGES: ISO-7 BIT CODE REPRESENTATION
(SENT AND RECEIVED WITH ATN=1)

BITS b_4 b_3 b_2 b_1	COL / ROW	0 0 0	MSG ①	0 0 1	MSG	0 1 0	MSG	0 1 1	MSG	1 0 0	MSG	1 0 1	MSG	1 1 0	MSG	1 1 1	MSG
0 0 0 0	0	NUL		DLE		SP		0		@		P		`		p	
0 0 0 1	1	SOH	GTL	DC1	LLO	!		1		A		Q		a		q	
0 0 1 0	2	STX		DC2		"		2		B		R		b		r	
0 0 1 1	3	ETX		DC3		#		3		C		S		c		s	
0 1 0 0	4	EOT	SDC	DC4	DCL	$		4		D		T		d		t	
0 1 0 1	5	ENQ	PPC③	NAK	PPU	%		5		E		U		e		u	
0 1 1 0	6	ACK		SYN		&		6		F		V		f		v	
0 1 1 1	7	BEL		ETB		'		7		G		W		g		w	
1 0 0 0	8	BS	GET	CAN	SPE	(		8		H		X		h		x	
1 0 0 1	9	HT	TCT	EM	SPD	)		9		I		Y		i		y	
1 0 1 0	10	LF		SUB		*		:		J		Z		j		z	
1 0 1 1	11	VT		ESC		+		;		K		[		k		{	
1 1 0 0	12	FF		FS		,		<		L		\		l		|	
1 1 0 1	13	CR		GS		-		=		M		]		m		}	
1 1 1 0	14	SO		RS		.		>		N		^		n		~	
1 1 1 1	15	SI		US		/		?	UNL	O		_	UNT	o		DEL	

MLA ASSIGNED TO DEVICE ④ — MTA ASSIGNED TO DEVICE — MEANING DEFINED BY PCG CODE

ADDRESSED COMMAND GROUP (ACG) — UNIVERSAL COMMAND GROUP (UCG) — LISTEN ADDRESS GROUP (LAG) — TALK ADDRESS GROUP (TAG) — SECONDARY COMMAND GROUP (SCG)

PRIMARY COMMAND GROUP (PCG)

NOTES: ① MSG = INTERFACE MESSAGE
② b_1 = DIO1 ... b_7 = DIO7
③ REQUIRES SECONDARY COMMAND
④ DENSE SUBSET (COLUMN 2 THROUGH 5)

Fig. 8. IEEE 488 command set reproduced from the IEEE Standard 488-1975/ANSI MC 1.1-1975, *p. 77.*

In many cases, a device will have a secondary address, which permits more than 31 effective addresses on the bus. For example, the Commodore printer might be set as device 4. To control internal functions, secondary addresses select the function in use. (See Commodore's "PET Communication with the Outside World," p. 19.) If a secondary address is in use, it is sent immediately after the talk or listen address, known as the primary address, with ATN true.

Several of the bus-management lines, such as SRQ, EOI, REN and IFC, serve special functions. Many instruments do respond to these, and often the response depends upon the instrument.

When ATN is low, about half the ASCII code is devoted to special commands, which come in defined sequences whose definition takes about two-thirds of the formal IEEE 488 specification. Most instruments use only a few of these.

Flipping Bits

The PET ultimately communicates to the rest of the world by the screen and some interface chips—two 6520s and one 6522. (For the specs on these chips, contact MOS Technology.) The 6520 and 6522 chips can only drive one TTL load, so the PET's IEEE lines are connected to some buffer chips to provide the currents needed in the IEEE 488 bus.

Table 1 indicates all of the PET's I/O line assignments as a reference. The PET utilizes all 60 I/O lines as shown here. Most of the IEEE lines are buffered with MC 3446P bidirectional line driver chips to provide the IEEE current requirements. SRQ is an input only and connects directly to the 6520 chip. IFC is buffered with a NAND and some resistors to the IEEE specification.

Table 1 reveals some interesting irregularities concerning the IEEE 488 bus: If EOI is true, the PET's display is turned off. (Programs that PEEK and POKE the display area in memory can use this to avoid snow.) Later-model PETs don't have this problem. REN isn't listed; the PET's REN line is wired to ground (true). IFC is not shown. The PET's IFC is connected to the power-on one-shot, which sets IFC true for about 100 ms when the PET is turned on. If you reset the PET by grounding the RES line, IFC may not go true. A better approach is to trigger the power-on one-shot by inserting a switch between power and the 555's power pin. The SRQ line is an input only. The PET's firmware does not use SRQ, so you have to program it directly.

In a 650x-based system, all I/O is seen as a set of memory addresses. This means that BASIC's PEEK and POKE can be used to control the IEEE 488 lines. Table 2 indicates the addresses and bits involved for the PET's IEEE lines. In most cases, a direct PEEK or POKE will do. Two lines, ATN in and SRQ in, require a more complex sequence. These are connected to CA1 and CB1 of a 6520, which set flag bits in the Interrupt Flag register. Resetting these bits requires a memory access to the DIO data register.

Table 3 lists the specific PEEKs and POKEs to individually sense or modify the IEEE lines. In many cases the PEEK or POKE values can be ANDed or ORed together to do several operations at once. If you have built the IEEE Blinkin Lites, try a

```
KEYBOARD PIA (6520)                ADDRESS: $ E810     59408

PAØ  Keyboard Row Select, LSB      PBØ  Keyboard Column A
PA1     "      "     "             PB1     "       "     B
PA2     "      "     "             PB2     "       "     C
PA3     "      "     "   , MSB     PB3     "       "     D
PA4  Switch, Cassette #1           PB4     "       "     E
PA5     "         "    #2          PB5     "       "     F
PA6  EOI In                        PB6     "       "     G
PA7  Diagnostic Jumper             PB7     "       "     H

CA1  Read, Cassette #1             CB1  Video Horiz Sync In
CA2  Screen Blank & EOI Out        CB2  Motor, Cassette #1

The Diagnostic LED will light if PAØ-High, PA1-High, PA2-Low, PA3-High.
```

```
USER PORT VIA (6522)               ADDRESS: $ E840     59456

PAØ  User Port LSB                 PBØ  NDAC In
PA1     "    "                     PB1  NRFD Out
PA2     "    "                     PB2  ATN Out
PA3     "    "                     PB3  Write, Both Cassettes
PA4     "    "                     PB4  Motor, Cassette # 2
PA5     "    "                     PB5  Video Horiz Sync In
PA6     "    "                     PB6  NRFD In
PA7     "    "  MSB                PB7  DAV In

CA1  User Port Handshake           CB1  Read, Cassette #2
CA2  Characters ROM Select         CB2  User Port Handshake

CA2 selects the MSB of the characters ROM, selecting the PET's
graphics or lower case characters for the display.
```

few of these PEEKs and POKEs to see how they work.

When I was flipping bits with PEEK and POKE for the IEEE lines, I was confused each time I had to figure out the decimal numbers for each changed bit. Perhaps it would be easier to display a byte of memory on the PET's screen in a "front panel" format with simulated LEDs for each bit and some simple keyboard commands to change bits and addresses. Memory Monitor (see Listing 1) does this.

When Memory Monitor is loaded and run, and the first page of instructions is read, the display in Fig. 9 is shown. A box with four parts appears in the middle of the screen with the title Memory Monitor placed above the box. Left of the box is a marker, >>, which indicates the part of the box accessible by the keyboard.

The top of the box shows the address of a memory location in decimal. If you press SPACE, the address will be erased, and a new number can be entered. Pressing number keys enters a new address, and a reverse-field cursor appears.

When a cursor isn't on the screen, pressing RETURN will move the marker to the next part of the box. (The second part in the box indicates the bit numbers and is skipped by the marker.)

The third part of the box displays a front panel made of solid or hollow "balls" (or "LEDs"). This shows the eight bits of the byte under investigation. The numbers above the "LEDs" indicate the bit numbers, 7 the MSB and 0 the LSB. To change the byte, enter 0 or 1 (or Shift-Q and Shift-W), and the cursor will appear. Pressing RETURN enters the value.

The fourth part of the box is the value of the byte in decimal and is entered in the same way

Listing 1. Memory Monitor.

```
10 PRINT"clr sp sp sp sp sp sp sp --> MEMORY MONITOR <--
20 PRINT"dn sp sp THIS PGM DISPLAYS A LOCATION IN THE
30 PRINT"PET'S MEMORY IN BOTH DECIMAL AND IN A
40 PRINT"'FRONT PANEL' FORMAT.
50 PRINT"dn sp sp YOU CAN CHANGE THE ADDRESS OR VALUE
60 PRINT"BY ENTERING A NEW VALUE WHEN THE '>>'
70 PRINT"MARKER IS NEXT TO THE ITEM YOU ARE
80 PRINT"CHANGING.
90 PRINT"dn sp sp PRESS 'RETURN' TO ENTER THE CHANGE
100 PRINT"OR TO MOVE THE MARKER.
110 PRINT"dn sp sp THE PGM CONSTANTLY PEEKS THE LOCATION
120 PRINT"WHEN YOU AREN'T CHANGING A VALUE. IF
130 PRINT"YOU CHANGE THE ADDRESS, THE PGM
140 PRINT"WILL SHOW THE NEW VALUE. IF YOU CHANGE
150 PRINT"A VALUE, IT IS POKED INTO MEMORY.
160 PRINT"dn sp sp 'H' WILL GIVE YOU SOME HELP FOR EACH
170 PRINT"ITEM.
190 PRINT"dn PRESS ANY KEY TO START
195 GETA$:IFA$=""THEN195

200 REM DRAW DISPLAY FORMAT
210 PRINT"clr dn dn dn dn dn";
220 D1$="@ @ @ @ @ @ @ @ @ @ @ @ @ @ @"      (15 shift-@)
230 D3$="rt rt rt rt rt rt rt rt rt rt rt"     (10 rt's)
240 PRINT"rt rt rt rt rt rt rt rt -->sp MEMORY MONITOR sp<--"
250 PRINTD3$"@"D1$"."
255 PRINTD3$"] ADDRESS:sp sp sp sp sp sp sp sp ]"  (] is a vertical line)
260 PRINTD3$"+"D1$"3"
265 PRINTD3$"] 7 sp 6 sp 5 sp 4 sp 3 sp 2 sp 1 sp Ø ]"
270 PRINTD3$"+ @ 2 @ 2 @ 2 @ 2 @ 2 @ 2 @ 2 @ 3"
280 PRINTD3$"] sp ] sp ] sp ] sp ] sp ] sp ] sp ] sp ]"
290 PRINTD3$"+ @ 2 @ 2 @ 2 @ 2 @ 2 @ 2 @ 2 @ 3"
300 PRINTD3$"] DECIMAL: sp sp sp sp sp sp sp ]"
310 PRINTD3$"-"D1$"-"
320 PRINT"dn dn dn"

    NOTE: For Lines 200-320 see Fig. 9.

400 REM IDLING PROGRAM
410 AD=59471:PT=1
500 REM DISPLAY ADDRESS
510 GOSUB 1000
520 REM DISP PANEL LITES
525 DT=PEEK(AD)
530 GOSUB 1200
540 REM DISP DECIMAL
550 GOSUB 1400
560 REM DISP PTR
570 GOSUB 1600
580 REM GET CHAR
590 GET A$
600 IF A$="" THEN 500
610 IF A$=CHR$(13) THEN 700
620 FG=Ø:GOSUB 2500
630 IF FG=Ø THEN 510
640 GOTO 210
700 REM BUMP PTR
710 GOSUB 1800
720 GOTO 510

1000 REM DISP ADDR
1010 PRINT"hm dn dn dn dn dn dn dn"
1020 V$=STR$(AD)+"sp sp sp sp sp sp sp"
1030 V$=MID$(V$,2,6)
1040 PRINT TAB(2Ø);V$
1050 RETURN

1200 REM DISP PANEL
1210 PRINT"dn dn dn"TAB(11);
1220 VT=DT:DV=128
1230 FOR J=1 TO 8
1240 IF VT/DV<1 THEN 1260
1250 PRINT" Q rt";:VT=VT-DV:GOTO 1300
1260 PRINT" W rt";
1300 DV=DV/2
1310 NEXT J

1400 REM DISP DECIMAL
1410 PRINT"dn"
1420 V$=STR$(DT)+" sp sp sp sp sp sp sp sp "
1430 V$=MID$(V4,2,6)
1440 PRINT TAB(2Ø);V$
1450 RETURN

1600 REM DISP PTR
1610 PRINT"hm dn dn dn dn dn dn dn"
1620 IF PT>1 THEN 1640
1630 PRINT TAB(8)">>";:RETURN
1640 PRINT"dn dn dn"
1650 IF PT>2 THEN 1670
1660 GOTO 1630
1670 PRINT"dn":GOTO 1630

1800 REM BUMP PTR
1810 GOSUB 1600
1820 PRINT"rt rt sp sp"
1830 PT=PT+1: IF PT>3 THEN PT=1
1840 GOSUB 1600
1850 RETURN

2000 RETURN              (This line probably isn't needed)

2500 REM CHANGE ITEM
2510 ON PT GOSUB 3000,3500,4000
2520 RETURN

3000 REM CHANGE ADDR
3010 IF A$="H" THEN GOSUB 4500:RETURN
```

VALUES FOR INPUTS			
IEEE LINE	ADDRESS (HEX)	ADDRESS (DECIMAL)	BIT
DIO 1	E820	59424	0
DIO 2	E820	59424	1
DIO 3	E820	59424	2
DIO 4	E820	59424	3
DIO 5	E820	59424	4
DIO 6	E820	59424	5
DIO 7	E820	59424	6
DIO 8	E820	59424	7
EOI	E810	59408	6
IFC	----	-----	-
SRQ	E823	59427	7
REN	----	-----	-
ATN	E821	59425	7
DAV	E840	59456	7
NRFD	E840	59456	6
NDAC	E840	59456	0

VALUES FOR OUTPUTS			
IEEE LINE	ADDRESS (HEX)	ADDRESS (DECIMAL)	BIT
DIO 1	E822	59426	0
DIO 2	E822	59426	1
DIO 3	E822	59426	2
DIO 4	E822	59426	3
DIO 5	E822	59426	4
DIO 6	E822	59426	5
DIO 7	E822	59426	6
DIO 8	E822	59426	7
EOI	E811	59409	3
IFC	----	-----	-
SRQ	----	-----	-
REN	----	-----	-
ATN	E840	59456	2
DAV	E823	59427	3
NRFD	E840	59456	1
NDAC	E821	59425	3

Table. 2 Addresses and bits for the IEEE 488 lines.

as the address.

If you press RETURN several times, the marker rotates through the three accessible parts of the box. To recall how to enter a value, press the letter H, which clears the screen and provides instructions.

The Memory Monitor eased the tedium and frustration of checking the PEEKs and POKEs used in the IEEE 488 memory locations. I have made Memory Monitor simple to use, and I consider it a good example of user-oriented programming.

Doing It the Hard Way

With direct access to the PET's IEEE 488 lines, you can use PEEK and POKE to operate an IEEE instrument "by hand." This is probably more difficult than using the IEEE Blinkin .ites box to communicate switch by switch because it takes more keystrokes to change a bit with POKE.

The next step is to write a BASIC program that performs the required IEEE 488 operations directly. Though the PET has these "built in," there are a few advantages to doing the whole thing in BASIC.

Everything goes slowly. As events happen, there is a chance of seeing them as they go by.

BASIC is accessible. If the PET or your instrument decides that the sky's the limit, pressing the STOP key can illuminate where the difficulties lie. The PET's built-in IEEE 488 services

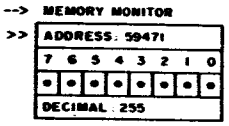

Fig. 9. Listing 1's initial display.

```
3020 V1=AD
3030 GOSUB 5000
3040 IF V2<0 THEN RETURN
3050 IF V2>65535 THEN RETURN
3060 AD=V2:RETURN

3500 REM CHANGE BINARY VALUE
3510 IF A$="H"THEN GOSUB 4600: RETURN
3520 V1=DT
3530 GOSUB 5500
3540 IF V2<0 THEN RETURN
3550 IF V2>255 THEN RETURN
3560 DT=V2:POKE AD,DT:RETURN

4000 REM CHANGE VALUE
4010 IF A$="H" THEN GOSUB 4500: RETURN
4020 V1=DT
4030 GOSUB 5000
4040 IF V2<0 THEN RETURN
4050 IF V2>255 THEN RETURN
4060 DT=V2:POKE AD,DT: RETURN

4500 PRINT"clr sp sp TYPE IN THE NEW NUMBER AND PRESS
4505 FG=1
4510 PRINT"RETURN. PRESS 'X' TO ABORT & NOT MAKE
4515 PRINT"THE CHANGE.
4520 PRINT" sp sp PRESS SPACE TO ERASE REST OF NUMBER.
4530 PRINT"dn sp sp PRESS ANY KEY
4540 GETA$: IFA$=""THEN 4540
4550 RETURN

4600 PRINT"clr sp sp ENTER '1' OR ' Q ' TO SET A BIT, AND
4610 PRINT"'0' OR ' W ' TO RESET A BIT. PRESS
4620 PRINT"RETURN WHEN DOEN.
4625 PRINT" sp sp PRESS SPACE TO SKIP A BIT.
4630 PRINT"dn sp sp PRESS ANY KEY
4640 GETA$:IF A$="" THEN 4540
4650 RETURN

5000 REM NUMERIC ENTRY
5010 REM POS CURSOR
5020 PRINT TAB(20);
5030 REM MAKE DISP STR
5040 D$=MID$(STR$(V1),2)+"sp sp sp sp sp sp sp"
5050 D$=LEFT$(D$,6)
5060 REM SET RVS PTR & RETURN VALUE
5070 PC=1:V2=-1
5080 REM SEE INPUT & ACT
5090 IF A$="X" THEN RETURN
5100 IF A$=CHR$(13) THEN V2=VAL(D$):RETURN
5110 IF A$<>" sp " THEN 5120
5112 IF PC=1 THEN D$="sp sp sp sp sp sp":GOTO 5210
5114 D$=LEFT$(D4,PC-1)+"sp sp sp sp sp sp":D$=LEFT$(D$,6)
5118 GOTO 5210
5120 IF A$<"0" OR A$>"9" THEN 5210
5125 REM REMAKE STRING
5130 DX$=D$:D$=""
5140 FOR J=1 TO 6
5150 IF PC=J THEN D$=D$+A$:GOTO 5170
5160 D$=D$+MID$(DX$,J,1)
5170 NEXT J
5180 PC=PC+1: IF PC>7 THEN PC=1

5200 REM DISPLAY RESULT & RESTORE CURSOR
5210 FOR J=1 TO 6
5220 IF J=PC THEN PRINT "rvs";
5230 PRINT MID$(D$,J,1);
5240 IF J=PC THEN PRINT "off";
5250 NEXT J:PRINT"lft lft lft lft lft lft";
5260 GET A$: IF A$="" THEN 5260
5270 GOTO 5090

5500 REM BINARY ENTRY
5510 PRINT TAB(11);
5520 FOR J= 1 TO 8
5525 V1=V1/2
5530 IF V1=INT(V1) THEN D$=" W "+D$: GOTO 5540
5535 D$=" Q "+D$
5540 V1 = INT(V1)
5550 NEXT J
5570 REM SET RVS PTR
5580 PC=1:V2=-1
5590 REM LOOK AT INPUT
5600 IF A$="X" THEN RETURN
5605 IF A$=CHR$(13)THEN 5780
5610 IF A$="sp" THEN 5715
5620 IF A$="1" OR A$=" Q " THEN A$=" Q ":GOTO 5660
5630 IF A$="0" OR A$=" W " THEN A$+" W ":GOTO 5660
5640 GETA$:IFA$=""THEN 5640
5650 GOTO 5600
5660 REM REMAKE STRING
5670 DX$=D$:D$=""
5680 FOR J= 1 TO 8
5690 IF PC=J THEN D$=D$+A$: GOTO 5710
5700 D$=D$+MID$(DX$,J,1)
5710 NEXT J
5715 PC=PC+1:IF PC>8 THEN PC=1
5720 REM DISP & FIX CURSOR
5730 FOR J= 1 TO 8
5735 IF J = PC THEN PRINT "rvs";
5740 PRINT MID$(D$,J,1)"rt";
5745 IF J=PC THEN PRINT "off";
5750 NEXT J:PRINT" lft lft lft lft lft lft lft lft
                lft lft lft lft lft lft lft lft";  (16 lft's)
5760 GOTO 5640
5770 REM MAKE VALUE
5780 V2=0:FORJ=1 TO 8
5785 V2=V2*2
5790 IF MID$(D$,J,1)=" W " THEN 5810
5800 V2=V2+1
5810 NEXT J
5820 RETURN
```

are mostly invisible, and there's often no way to find out why something went wrong.

Everything is under control. It is simple enough to display every step with suitable messages to the screen. If necessary, you can insert a GET loop to make the PET wait until a key is pressed before proceeding.

Changes are easy.

It's an educational experience—those who must learn the "nuts and bolts" of the IEEE bus will find a BASIC emulator useful.

I constructed the BASIC 488 program (see Listing 2) to provide the following essential services: put the PEEK and POKE values into variable form for reasonably fast execution and to simplify debugging with direct commands; do most of the PEEKs and POKEs for line control as short subroutines; provide the listen and talk handshake sequences for one byte and display their progress; provide a way to send and receive strings to a device on the bus; set the program up as a skeleton onto which you can add specific programs to suit changing needs.

Table 4 indicates the subroutines and variables used in the BASIC 488 program. Load these subroutines and then add the code you need for your devices. Some devices, such as those by Commodore, may not follow the IEEE time standard, and the BASIC 488 program will not be fast enough to prevent time-outs.

I built the program from the bottom up, starting with subroutines 1500 and the series starting at 9000. Subroutine 1500 sets up the essential variables. A1-7 are the addresses of the PEEK/POKE locations; M0-M7 and N0-N8 are AND and OR masks to extract bits 0-7 from a location (or to set the desired bits); 01-07 are the original values for addresses A1-A7. (POKE A1,01, for example, will restore location A1 to the PET's power-on value, which helps you to recover from disasters.)

The variables H1 to H6 are the sense values for the IEEE lines. For example, if H1 is 1, the DAV line is true. If H1 is zero, DAV is false.

When you enter BASIC 488, enter lines 1000-1620 and lines 9000-9640 first. Use the IEEE Blinkin Lites to check that the subroutines in the 9000 series function correctly. First, GOSUB 1000 in direct mode to set things up. Then, GOSUB to the section under test and look at the Blinkin Lites to see what happened. A PRINT H1 will inform you of the sensing subroutines' results. Be sure to thoroughly test the 9000 series first!

All DIO Lines:*

IN:	POKE 59426,255:V = PEEK(59424):V = (NOT(V))AND255
OUT.	V = (NOT(V))AND255: POKE 59426,V

EOI

IN:	V = 1:IF PEEK(59408)AND 64 THEN V = 0
TRUE OUT:	POKE 59409, PEEK(59409) AND 247
FALSE OUT:	POKE 59409, PEEK(59409) OR 8

REN & IFC – Not Applicable

SRQ**

IN:	V = 0:IF PEEK(59427) AND 128 THEN V = 1
	Z = PEEK(59426)
LO-HI	POKE 59427, PEEK(59427) OR 2
HI-LO	POKE 59427, PEEK(59427) AND 253

ATN**

IN:	V = 0:IF PEEK(59425) AND 64 THEN V = 1
	Z = PEEK(59424)
LO-HI:	POKE 59409, PEEK(59409) OR 2
HI-LO:	POKE 59409, PEEK(59409) AND 253
TRUE OUT:	POKE 59456, PEEK(59456) AND 251
FALSE OUT:	POKE 59456, PEEK(59456) OR 4

DAV

IN:	V = 1: IF PEEK(59456) AND 128 THEN V = 0
TRUE OUT:	POKE 59427, PEEK(59427) AND 247
FALSE OUT:	POKE 59427, PEEK(59427) OR 8

NRFD

IN:	V = 1: IF PEEK(59456) AND 64 THEN V = 0
TRUE OUT:	POKE 59456, PEEK(59456) AND 253
FALSE OUT:	POKE 59456, PEEK(59456) OR 2

NDAC

IN:	V = 1: IF PEEK(59456) AND 1 THEN V = 0
TRUE OUT:	POKE 59425, PEEK(59425) AND 247
FALSE OUT:	POKE 59425, PEEK(59425) OR 8

*The extra parenthesis in the complementation of V is required, for the PET evaluates AND before NOT.

**The HI-LO or LO-HI determines which transition the CA/CB1 inputs will respond to. Set the HI-LO or LO-HI before doing the IN: line. The Z = PEEK resets the flag bit. Be sure to reset the flag bit before checking the first time.

SRQ OUT is not available on the PET.

Table 3. PEEKs and POKEs for the IEEE 488 lines.

Listing 2. BASIC 488 program.

```
1000 REM **** IEEE 488 ****
1005 REM   GREGORY YOB, JAN 1979
1010 REM   BOX 354, PALO ALTO CA 94301
1015 REM
1020 REM THESE ROUTINES PERMIT DIRECT
1025 REM MANIPULATION OF THE PET IEEE
1030 REM 488 BUSS LINES AND (SLOW!)
1035 REM IEEE 488 COMMAND AND DATA
1040 REM TRANSFERS
1045 REM
1500 REM -- INITIALIZATION --
1510 RESTOR:READ A1,A2,A3,A4,A5,A6,A7
1520 DATA 59424,59426,59425,59427,59408,59456,59409
1530 READ M0,M1,M2,M3,M4,M5,M6,M7
1540 DATA 1,2,4,8,16,32,64,128
1550 READ N0,N1,N2,N3,N4,N5,N6,N7
1560 DATA 254,253,251,247,239,223,191,127
1570 READ N8
1580 DATA 255
1590 READ O1,O2,O3,O4,O5,O6,O7      (Each of these is Letter O)
1600 DATA 255,266,60,60,249,255,60
1610 DEF FNF(X)=(NOT(X))AND255
1620 RETURN

7000 PRINT"clr GET MESSAGE"
7010 PRINT"dn PRESS KEY TO START"
7020 GETA$:IFA$=""THEN RETURN        ("" is an empty string)
7030 D2=FNF(DV+64):GOSUB9450:GOSUB8500:GOSUB9470
7040 B$=""
7050 GOSUB 8000:IF FNF(D1)=13THEN7070
7060 B$=B$+CHR$(FNF(D1)):GOTO7050
7070 GOSUB8000:REM LF BUCKET
7080 PRINT"dn dn MESSAGE IS: sp"B$
7090 RETURN

7500 PRINT"clr SEND MESSAGE"
7510 INPUT"dn dn MESSAGE:";C$
7520 D2=FNF(DV+32):GOSUB9450:GOSUB8500:GOSUB9470
7530 FOR J=1 TO LEN(C$)
7540 D2=FNF(ASC(MID$(C$,J)))
7550 GOSUB8500:NEXTJ
7560 PRINT"dn dn MESSAGE SENT: sp"C$
7570 RETURN

8000 PRINT"clr LISTEN HANDSHAKE dn"
8010 GOSUB9350:GOSUB9250:GOSUB9370
8020 PRINT" sp NRFD TRUE dn":PRINT" sp NDAC TRUE"
     :PRINT" sp NRFD FALSE"
8030 PRINT"WAITING FOR DAV TRUE"
8040 GETA$:IFA$<>""THENPRINT"--FORCED":GOTO8060
8050 GOSUB9100:IFH1=0THEN8040
8060 GOSUB9000:PRINT"dn spDATA:"FNF(D1)CHR$(FNF(D1))
8070 GOSUB9350:GOSUB9270
8080 PRINT"dn sp NRFD TRUE":PRINT" sp NDAC FALSE"
8090 PRINT"WAITING FOR DAV FALSE"
8100 GETA$:IFA$<>""THENPRINT"--FORCED":GOTO8120
8110 GOSUB9100:IFH1=1THEN8100
8120 GOSUB9250
8130 PRINT"dn sp NDAC TRUE"
8140 RETURN

8500 PRINT"clr TALK HANDSHAKE"
8510 GOSUB9170
8520 PRINT"dn sp DAV FALSE"
8530 GOSUB9200:GOSUB9300
8540 IF H1+H2>0 THEN 8570
8550 PRINT"dn>> sp ERROR STATE-PRESS KEY TO FORCE"
8555 PRINT"NOTE: MAKE NRFD, NDAC TRUE"
8560 GETA$;IFA$=""THEN8560
8570 GOSUB9050
8580 PRINT"dn DATA ON LINE:"FNF(D2)CHR$(FNF(D2))
```

Nothing else will work if these don't!

If all else fails, refer to Tables 1, 2 and 3 and try a few direct PEEKs and POKEs to ensure that the IEEE lines are functional.

Add lines 8000-8140 and lines 8500-8690, which you can check by attaching the 488 Blinkin Lites and carefully tracing through the handshake flowchart in Fig. 7. Again, it is essential to be sure these routines work correctly. An additional benefit is that you will learn the handshake sequence in detail.

Note that the data transferred, D1 or D2, must be complemented with the FNF function as it enters or leaves the IEEE bus. In some of the waiting loops, such as lines 8030-8050, a GET A$ check is inserted. If the instrument hangs up, pressing a key will force the handshake to proceed, and a suitable message will appear on the screen. As the handshakes proceed, their progress is reported to the screen for your reference.

Next, add lines 7000-7570. These routines require a device address, DV, to function correctly. Subroutine 7000 will fetch a message from a device, and subroutine 7500 will send a message. The strings B$ and C$ are used to store the messages.

Most devices will send an EOI along with the last character of their messages. This will turn off the screen. In some cases, you will have to provide an EOI, which will again turn off the screen. To recover, enter:

GOSUB 9570 (and RETURN)

Another approach is to move the cursor down until the screen scrolls. A scroll turns the screen off, and then on. If you have a 16K PET, the screen will not blink.

Testing the last part via the IEEE Blinkin Lites is tedious. If you have an instrument available, try talking to it! Be sure you know *exactly* what your instrument expects and its responses!

Talking to the HP Clock via BASIC 488

Now that you have checked out BASIC 488 by hand, try it with a real live instrument! I connected the HP clock, loaded BASIC 488 and gave it a try (see Example 1). The clock's front panel shows the reset worked.

These commands can be compressed to one line (see Example 2).

Next, try to read the clock. Address the clock to talk, then read the 14-character message shown in Example 3. If you look at the line DATA: on the display for the Listen Handshake, you can barely see the clock's message. A different version (see Example 4) will pick up the message and leave it later. Below the Listen Handshake display appears the clock's message:
0101000520

The BASIC 488 program has two routines for sending and reading entire strings via the IEEE 488. Subroutine 7000 addresses device DV to talk and read a string. Subroutine 7500 addresses device DV to listen and sends a string. (Note: Routine 7000 reads a string until a carriage return is seen, and then reads one more character. This is because the HP clock ends messages with CR and LF. You might have to change this for your device.)

To reset the clock:
DV = 7:GOSUB 7500
The screen clears and asks for

```
8590 print"dn WAITING FOR NRFD FALSE"
8600 GETA$:IFA$<>""THENPRINT"--FORCED":GOTO8620
8610 GOSUB9300:IFH3=1THEN8600
8620 GOSUB9150.
8630 PRINT"dn sp DAV TRUE"
8640 PRINT"WAITING FOR NDAC FALSE"
8650 GETA$:IFA$<>""THEN8670
8670 GOSUB9170
8680 PRINT"dn sp DAV FALSE"
8690 RETURN

9000 POKEA2,N8:D1=PEEK(A1):RETURN
9050 POKEA2,D2:RETURN
9100 H1=1:IFPEEK(A6)ANDM7THENH1=Ø
9110 RETURN
9150 POKEA4,PEEK(A4)ANDN3:RETURN
9170 POKEA4,PEEK(A4)ORM3:RETURN
9200 H2=1:IFPEEK(A6)ANDMØTHENH2=Ø
9210 RETURN
9250 POKEA3,PEEK(A3)ANDN3:RETURN
9270 POKEA3,PEEK(A3)ORM3:RETURN
9300 H3=1:IFPEEK(A6)ANDM6THENH3=Ø
9310 RETURN
9350 POKEA6,PEEK(A6)ANDN1:RETURN
9370 POKEA6,PEEK(A6)ORM1:RETURN
9400 PRINT"NO ATN LEVEL":STOP
9430 H4=Ø:IFPEEK(A3)ANDM7THENH4=1
9440 ZZ=PEEK(A1):RETURN
9450 POKEA6,PEEK(A6)ANDN2:RETURN
9470 POKEA6,PEEK(A6)ORM2:RETURN
9500 H5=1:IFPEEK(A5)ANDM6THENH5=Ø
9510 RETURN
9550 POKEA7,PEEK(A7)ANDN3:RETURN
9570 POKEA7,PEEK(A7)ORM3:RETURN
9600 REM SRQ NOT OUTPUT
9630 H6=Ø:IFPEEK(A4)ANDM7THENH6=1
9640 ZZ=PEEK(A2):RETURN
```

Entry Points:

SUBROUTINE 1500	Initialization (Must be done first)
SUBROUTINE 7000	Get Message as B$, Requires DV
SUBROUTINE 7500	Put Message C$, Requires DV
SUBROUTINE 8000	Listen Handshake
SUBROUTINE 8500	Talk Handshake
SUBROUTINES 9000ff19600	IEEE Lines Primitives
9000	Read DIO as D1
9050	Write DIO as D2
9100	Read DAV as H1
9150	Set DAV TRUE
9170	Set DAV FALSE
9200	Read NDAC as H2
9250	Set NDAC TRUE
9270	Set NDAC FALSE
9300	Read NRFD as H3
9350	Set NRFD TRUE
9370	Set NRFD FALSE
9400	Trap for ATN
9430	Check ATN as H4 (If changed)
9450	Set ATN TRUE
9470	Set ATN FALSE
9500	Read EOI as H5
9550	Set EOI TRUE (Screen will blank)
9570	Set EOI FALSE (Screen returns)
9630	Check SRQ as H6 (If changed)

Variables:

PEEK/POKE ADDRESSES		ORIGINAL VALUES	
A1	59424	01	255
A2	59426	02	255
A3	59425	03	60
A4	59427	04	60
A5	59408	05	249
A6	59456	06	255
A7	59409	07	60

Masks:

M0	0000 0001	1	N0	1111 1110	254
M1	0000 0010	2	N1	1111 1101	253
M2	0000 0100	4	N2	1111 1011	251
M3	0000 1000	8	N3	1111 0111	247
M4	0001 0000	16	N4	1110 1111	239
M5	0010 0000	32	N5	1101 1111	223
M6	0100 0000	64	N6	1011 1111	191
M7	1000 0000	128	N7	0111 1111	127
			N8	1111 1111	255

Miscellaneous:

DV	Device Address
A$	Keyboard dummy entry
B$	Message from Device
C$	Message to Device

Functions:

FNF(X)	Returns complement of argument

Table 4. BASIC 488 program notes.

```
CLR
GOSUB 1500:GOSUB 1900          Get everything ready . . .
PRINT FNF(32 + 7)
  216                          This is the value for D2 as a listen address.
GOSUB 9450                     Make ATN true.
D2 = 216:GOSUB 8500            Send listen address via handshake
TALK HANDSHAKE                 The PET responds with the step-by-step
  DAV FALSE                    output handshake and goes successfully
DATA ON LINE 39                through the entire process
WAITING FOR NRFD FALSE
  DAV TRUE
WAITING FOR NDAC FALSE         The HP Clock's "addressed" light turns on!
  DAV FALSE
READY.
GOSUB 9470                     Make ATN false
PRINT FNF(ASC("R"))            R resets the clock
  173
D2 = 173:GOSUB 8500            Send 'R' as data . . . . .
( . . . . . )                  And this handshakes through OK too.
```

Example 1. My dialogue with the HP clock via BASIC 488.

```
D2 = 216:GOSUB9450:GOSUB8500:GOSUB9470:D2 = 173:GOSUB8500
```

Example 2. A one-line command for Example 1.

```
PRINT FNF(64 + 7)              Find out D2 for talk address
 184
D2 = 184:GOSUB9450:GOSUB8500:GOSUB9470
( . . . . . )                  The handshake goes through
FOR J = 1 TO 14:GOSUB 8000:NEXT
( . . . . . for 14 times . . . . . )
```

Example 3. The dialogue for reading the clock.

the message (see Example 5).

The Talk Handshake flashes on the screen twice, and the message sent is displayed below:

```
( . . . . . )
MESSAGE SENT: R
```

The program uses routine 7000 to read the time. Since DV is already set, we don't have to reassign DV = 7 again. See Example 5. Note that there are three spaces between the colon and the first zero. Two of these are from the HP clock, which starts all messages with two blanks.

The BASIC 488 program, though slow to operate, never times-out and lets you control the IEEE 488 bus. This is helpful when you debug a new IEEE device with your PET.

If you are an experienced 6502 programmer, it is simple to translate the BASIC 488 program into a set of machine-language routines. If you do so, I'd like a copy (tape and source). Listing 3 shows a copy of the IEEE handshakes in machine language. (From the *PET User Notes*, PO Box 371, Montgomeryville, PA 18936, Vol. 1, Issue 7, (Nov.-Dec. '78), p. 8. This is a reprint from the Commodore PET Users Club of England.)

The PET handles the IEEE 488 as a file. Part 2 will cover this. ■

IEEE Bus Handshake Routine - Main Program

```
1800 A200    LDX #00     prepare index register
1802 A9FB    LDA #FB     set ATN low
1804 2D40E8  AND E840
1807 8D40E8  STA E840
180A A928    LDA #28     MLA (28 for this device)
180C 8501    STA 01
180E 208018  JSR 1880    handshake into bus
1811 A908    LDA #08     GET
1813 8501    STA 01
1815 208018  JSR 1880    handshake
1818 A948    LDA #48     MTA
181A 8501    STA 01
181C 208018  JSR 1880    handshake
181F A9FD    LDA #FD     set NRFD low
1821 2D40E8  AND E840       (ready to receive data)
1824 8D40E8  STA E840
1827 A9F7    LDA #F7     and NDAC low also
1829 2D21E8  AND E821
182C 8D21E8  STA E821
182F A904    LDA #04     set ATN high
1831 0D40E8  ORA E840
1834 8D40E8  STA E840
1837 A008    LDY #08     ready to count 8 bytes
1839 20B018  JSR 18B0    handshake data from bus
183C A502    LDA 02      result to A
183E 9D0119  STA 1901,X  store in 1901+X
1841 E8      INX
1842 88      DEY
1843 D0F4    BNE 1839    jump if Y not zero
1845 A9FB    LDA #FB     set ATN low
1847 2D40E8  AND E840
184A 8D40E8  STA E840
184D A902    LDA #02     set NRFD high
184F 0D40E8  ORA E840
1852 8D40E8  STA E840
1855 A908    LDA #08     set NDAC high
1857 0D21E8  ORA E821
185A 8D21E8  STA E821
185D A95F    LDA #5F     UNT
185F 8501    STA 01
1861 208018  JSR 1880    handshake to bus
1864 A904    LDA #04     set ATN high
1866 0D40E8  ORA E840
1869 8D40E8  STA E840
186C CE0019  DEC 1900    decrease counter
186F D091    BNE 1802    jump if not zero
1871 60      RTS         return to BASIC program
```

Subroutine to Handle Handshake Into Bus

```
1880 AD40E8  LDA E840    NRFD ?
1883 2940    AND #40
1885 F0F9    BEQ 1880    jump back if not ready
1887 A501    LDA 01      ready:  get data byte
1889 49FF    EOR #FF     complement it
188B 8D22E8  STA E822    send to bus
188E A9F7    LDA #F7     set DAV low
1890 2D23E8  AND E823
1893 8D23E8  STA E823
1896 AD40E8  LDA E840    NDAC ?
1899 2901    AND #01
189B F0F9    BEQ 1896    jump back if not accepted
189D A908    LDA #08     accepted;  set DAV high
189F 0D23E8  ORA E823
18A2 8D23E8  STA E823
18A5 A9FF    LDA #FF     255(10) into bus
18A7 8D22E8  STA E822
18AA 60      RTS         return to main
```

Subroutine to Handle Handshake From Bus

```
18B0 A902    LDA #02     set NRFD high
18B2 0D40E8  ORA E840
18B5 8D40E8  STA E840
18B8 AD40E8  LDA E840    DAV ?
18BB 2980    AND #80
18BD D0F9    BNE 18B8    jump back if not valid
18BF AD20E8  LDA E820    get data byte from bus
18C2 49FF    EOR #FF     complement
18C4 8502    STA 02      store in $ 0002
18C6 A9FD    LDA #FD     set NRFD low
18C8 2D40E8  AND E840
18CB 8D40E8  STA E840
18CE A908    LDA #08     set NDAC high
18D0 0D21E8  ORA E821
18D3 8D21E8  STA E821
18D6 AD40E8  LDA E840    DAV high ?
18D9 2980    AND #80
18DB F0F9    BEQ 18D6    jump back if not
18DD A9F7    LDA #F7     set NDAC low
18DF 2D21E8  AND E821
18E2 8D21E8  STA E821
18E5 A9FF    LDA #FF     255(10) into bus
18E7 8D22E8  STA E822
18EA 60      RTS         return to main
```

IEEE Bus Handshake Routine Object Listing

```
1800 A2 00 A9 FB 2D 40 E8 8D
1808 40 E8 A9 28 85 01 20 80
1810 18 A9 08 85 01 20 80 18
1818 A9 48 85 01 20 80 18 A9
1820 FD 2D 40 E8 8D 40 E8 A9
1828 F7 2D 21 E8 8D 21 E8 A9
1830 04 0D 40 E8 8D 40 E8 A0
1838 08 20 B0 18 A5 02 9D 01
1840 19 E8 88 D0 F4 A9 FB 2D
1848 40 E8 8D 40 E8 A9 02 0D
1850 40 E8 8D 40 E8 A9 08 0D
1858 21 E8 8D 21 E8 A9 5F 85
1860 01 20 80 18 A9 04 0D 40
1868 E8 8D 40 E8 CE 00 19 D0
1870 91 60 EA EA EA EA EA EA
1878 EA EA EA EA EA EA EA EA
1880 AD 40 E8 29 40 F0 F9 A5
1888 01 49 FF 8D 22 E8 A9 F7
1890 2D 23 E8 8D 23 E8 AD 40
1898 E8 29 01 F0 F9 A9 08 0D
18A0 23 E8 8D 23 E8 A9 FF 8D
18A8 22 E8 60 EA EA EA EA EA
18B0 A9 02 0D 40 E8 8D 40 E8
18B8 AD 40 E8 29 80 D0 F9 AD
18C0 20 E8 49 FF 85 02 A9 FD
18C8 2D 40 E8 8D 40 E8 A9 08
18D0 0D 21 E8 8D 21 E8 AD 40
18D8 E8 29 80 F0 F9 A9 F7 2D
18E0 21 E8 8D 21 E8 A9 FF 8D
18E8 22 E8 60

0001 data to go into bus
0002 data from bus

1900 counter for number of data transfers

1901 start of results area
```

Listing 3. IEEE bus handshake routine in machine language. MLA is My Listen Address; MTA is My Talk Address; UNT is Untalk Command.

Gregory Yob
Box 354
Palo Alto, CA 94301

Get Your PET on the IEEE 488 Bus

Part 2 of this "opus computerus" examines the file characteristics of the IEEE 488 bus.

Your PET has a "built-in" way of communicating through the IEEE 488 bus. In BASIC, the IEEE 488 looks like a file—just as the cassettes are files. The OPEN statement is used to specify a physical device number of 4 to 30, and the open logical file now talks via the IEEE 488 bus.

A complete understanding of PET tape files is a prerequisite for working with the IEEE 488 as a BASIC file. An article in the January 1979 *Kilobaud Microcomputing* ("PET Techniques Explained") covers many "innocent" errors that will result in mysterious malfunctions.

IEEE 488 Information Transfers

Talking to a Device.

1. OPEN a BASIC file to the device's address. For example, OPEN 1,4 will open the IEEE bus to device 4. Your BASIC program will see this as file #1.

2. PRINT# to your OPENed file. PRINT#1,"HELLO, DEVICE" will address the device to listen, send the string HELLO, DEVICE, add a carriage return with EOI true and then issue the UNT (Untalk) command.

3. Repeat step 2 as needed. Note that after each PRINT#, the IEEE bus is free, since the UNT has been sent.

PRINT# will send the same characters, including the skip character after numbers, as PRINT does to the screen. If you want to send several items, be sure that any needed delimiters, such as ",", are included.

Listening to a Device.

1. OPEN a BASIC file to the device's address.

2. Use INPUT# or GET# to fetch a line or a character from the IEEE bus.

3. Check the status word, ST, for an error, such as time-out. If the device is slow, the PET will complete the INPUT# or GET# and put a nonzero value into ST, which must be checked immediately after the I/O operation. If ST indicates a time-out, jump back to step 2.

4. Convert the data from the INPUT# or GET# as needed, and if more is needed, go to step 2.

Note that after each INPUT# or GET#, the UNT command is sent to the IEEE bus. This will truncate long messages from the device, especially with GET#. Also note that INPUT# (string) and GET# (string) work the best. The BASIC string functions (MID$, RIGHT$, LEFT$ and VAL) will help you get the data into a usable form.

Talking to More than One Device.

1. OPEN a file for each device.

2. Using CMD, send a dummy message to each device. For example, CMD 1:CMD 2:CMD 3 will set up each device (as specified in the OPENs for files 1, 2 and 3) by sending carriage returns to the devices and leaving them as listeners on the bus.

3. PRINT# to the IEEE bus. Any of the OPENed files may be used.

4. Repeat steps 2 and 3 as needed. Since PRINT# ends with the UNT, step 2 must be repeated after each PRINT#.

Transfer from One Device to Another.

1. OPEN a file for each device.

2. CMD to the device that is to be the listener.

3. INPUT# from the device that is to be the talker.

4. Repeat step 3 as needed.

INPUT# does not send a UNL, so the device that was CMDed remains on the bus as a listener. All information sent by the talker to the PET is also received by the listener. To turn off the listener, use a PRINT# to the listener's file. If the talker is slow, check ST and repeat step 3 as required.

LISTing a BASIC Program to a Device

1. OPEN a file to the device.

2. CMD to the device.

3. Enter the LIST command.

4. When the LIST is finished, do a CLR.

The PET's graphics and cursor characters will not print correctly on a standard ASCII printer. (I have a BASIC listing program available.)

The best way to learn the PET files and IEEE 488 is by specific examples. After a detour through CMD, we will continue with two examples. These should provide you with enough information to get started. If you have no success, refer to the section on Common Errors (found later in this installment).

CMD

CMD is an unusual PET command. Consider its functions:

1. Anything that BASIC wants to say is now routed to the device that CMD's file number refers to. If this isn't the screen, nothing that BASIC says will appear on the screen.

2. If a list of variables and literals is provided after the CMD, they will be sent to the device in the same way as PRINT# will.

3. However, if the device is on the IEEE bus, no UNL will be sent, so the device will remain in the listening state and receive any following data sent on the IEEE bus.

To see how CMD operates, get two scratch tapes and enter the program in Example 1. Now SAVE and VERIFY this program on one of your tapes. Put the other tape in the tape unit and execute the following:

```
OPEN 1,1,1
PRESS PLAY & RECORD ON TAPE#1
```

Perform this and wait until the tape stops.

```
OK
READY.
```

Now enter CMD 1. Note that READY. didn't appear; it was provided by BASIC and is now residing in the tape buffer. The cursor is blinking below the C in CMD. Continue with:

```
10  REM CMD EXAMPLE
20  PRINT""""
30  OPEN 1
40  GET#1, A$
50  PRINT A$;
60  IF A$ = CHR$(90)THEN PRINT"""":END
70  GOTO 40
80  REM Z
```

Example 1.

```
LIST
CLOSE 1
CLR
READY
```

Note that the CLOSE 1 didn't get the READY. back. It took the CLR to return BASIC's messages to the screen. If you enter LIST, the program will appear on the screen. Rewind the tape and RUN. Three asterisks now appear after the RUN. These were printed by the program. This is one reason I don't trust my PET after a CMD. The text between the OK and the ending READY was found as a data file.

When the PET was under the influence of CMD, the letters you typed in were put onto the screen. This echoing is done by the PET's operating system, so CMD won't put these out to the device.

Though CMD looks like a good way to LIST program to tapes as data files, there is a snag. My example is shorter than 191 characters, and a LIST via CMD isn't smart enough to "jiffy" the data tape (this has been fixed on the new PETs). You run the risk of losing tape records when you try to read an "unjiffied" tape.

Try to verify that CMD 1,"HELLO OUT THERE" will print HELLO OUT THERE onto the tape. Remember that if you CMD a device on the IEEE 488 bus, any PRINT# to the bus will require a repetition of the CMD if you want the device to remain in the listening state.

Talking to the Clock Again

(For a description of the HP clock see part 1 of this article.)

First, you must check the device address on the DIP switch (which will be near the 488 female connector) and make sure the address is in the range 4 to 15. The enter a short program (Example 2) into the PET. This program consists of three subroutines to facilitate communicating with the clock. Remember that the PET will not accept an INPUT statement as a direct command.

```
10  OPEN 1,7
20  RETURN
100 INPUT"SAY TO CLOCK:";S$
110 PRINT#1,S$
120 RETURN
200 INPUT#1,C$
210 PRINT"CLOCK SAYS: ";C$
220 RETURN
```

Example 2.

First, enter GOSUB 10 as a direct command. This opens file 1 to device 7, which is our clock on the IEEE bus. OPEN merely sets things up; nothing is sent to the bus yet.

To read the time, enter GOSUB 200:

```
GOSUB 200
CLOCK SAYS: 0103020204 (Jan. 3, 2:02:04 AM)
```

Your PET might give ? SYNTAX ERROR after this operation. This is a harmless feature of the PET.

To set the clock, using Jan. 29, 9:17 PM, as our example, enter:

```
GOSUB 100
SAY TO CLOCK? RDDDDDDDDDDDDDD
DDDDDDDDDDDD(28 Ds)
```

The clock starts at day 1. To set to day n, use n – 1 Ds. To set the hour, enter the following.

```
GOSUB 100
SAY TO CLOCK? HHHHHHHHHHHHHHH
HHHHHH(21Hs)
```

Minutes and seconds are set similarly.

```
GOSUB 100
SAY TO CLOCK? MMMMMMMMMMMM
MMMMSSS (17Ms, 3Ss)
```

We are now set to 9:17:03. When I did this by hand, the clock moved forward about a minute, so the number of M's used should be changed to accommodate for this.

Talking to the HP 8165A Programmable Signal Source

(For a description of the HP 8165A, see part 1 of this article.)

The 8165A is a fine instrument with many switches, knobs, buttons and options and a correspondingly wide array of IEEE 488 commands (see Fig. 12, part 1).

The precise contents of each example concern the 8165A, which is an instrument you will probably never meet! My intention is to show you how direct mode commands—that is, BASIC statements without line numbers—can be used to control an instrument and help in debugging.

First, I hooked the 8165 to the 488 cable, and the PET turned on. The 8165 was addressed to 8. When the PET came on, IFC was true for about one second. This put the 8165 in local mode, where the front panel works as usual. Many instruments will ignore their front panels when the 488 bus addresses them. Once the PET addresses the 8165, you cannot control it from the front panel anymore. (An LED indicates this on the 8165.)

The following short program takes care of input from the instrument:

```
10  INPUT#1, A$
20  PRINT A$
```

This substitutes for the illegal direct command (INPUT#1,A$: PRINTA$), which I would like to use, but the PET forbids (try it and see!).

Since I wanted the 8165 to output a 1 kHz sine wave at an amplitude of 1.5 volts, I used the following IEEE commands:

F1—Set to sine wave
FRQ 1 kHz—Set frequency
AMP 1.5 V—Set amplitude
I1—Set to normal operation (continuous signal output)

First, open the IEEE file:

```
OPEN 1,8
READY.
```

Then send the settings:

```
PRINT #1,"F1"  (At this point, the "Remote" LED went on, and I can no longer work the front panel.)
PRINT #1,"FRQ1KHZ"
PRINT #1,"AMP1.5V"
PRINT #1,"I1"
```

Nothing happened! My scope showed only a flat trace! Upon reviewing my steps, I noticed that I overlooked the Disable Output (OD) and Enable Output (OE) commands. I entered PRINT #1,"OE", and a sine wave appeared on the scope.

You could also send this setting as one string. For example, PRINT #1,"F2FRQ1.2KHZAMP 1.2VI1OE" sets up a 1.2 kHz triangle wave at 1.2 V amplitude.

The 8165 can also report some of its switch settings. Now we can use the tiny program in the PET:

```
GOTO 10
F1 D2 12 FM0 AM0
```

Since the PET has difficulty with GOSUB in direct mode and the IEEE bus, we must make a program change:

```
10  INPUT#1, A$
20  PRINT A$
30  RETURN
```

We will quickly be reminded that any time we change a program, all the variables, including opened files, will be lost:

```
GOSUB 10
?FILE NOT OPEN ERROR IN 10
```

So we try again:

```
OPEN 1,8
GOSUB 10
F1 D2 I2 FM0 AM0
?SYNTAX ERROR IN 22066
```

The PET will provide the ?SYNTAX ERROR about 90 percent of the time when the IEEE is accessed via the INPUT# statement and the PET is executing a directly called subroutine. However, this doesn't appear to affect anything. I avoided this by not making the little program a subroutine the first time.

So, if you are in a pinch, remember that the PET's direct command capability can rescue you with IEEE 488 devices and provides an inexpensive way to explore a new instrument.

Talking to More than One Device

Now that each of the instruments has been in the bus individually, the next step is to try the 488 with both of them on at the same time. I connected the HP clock and the 8165 to the 488 bus and gave the clock address #7, and the 8165 address #8. Then I entered the short program for INPUTs:

```
10  INPUT #1, A$
20  PRINT A$
30  END
100 INPUT #2, B$
110 PRINT B$
120 END
```

First, OPEN the files:

```
OPEN 1,7
OPEN 2,8
```

If you get a ?FILE OPEN ERROR, just enter CLR and start over.

Taking a peek at the clock resulted in:

```
GOTO 10
0130051957      (30 Jan., 5:19:57)
```

And peeking at the 8165 gets me:

```
GOTO 100
F1 D2 I2 FM0 AM0
```

which is the usual mystery message that the 8165 says to me. There isn't any point in explaining this message, for your instrument will say something different and meaningful only to you.

PRINT #1 and PRINT #2 will

```
X$ = "":FORJ = 1TO14:GOSUB8000:X$ = X$ + CHR$(FNF(D1)):NEXT:PRINTX$
0101000520
```

Example 4. Putting the clock's message into X$, and the contents of X$.

```
DV = 7:GOSUB7500
SEND MESSAGE

MESSAGE:? R                    R for reset

GET MESSAGE

PRESS KEY TO START

(.....A lot of Listen Handshakes....)

MESSAGE IS:  0101000158
```

Example 5. Resetting the clock.

Program Listing Conventions

The PET's graphics and cursor control characters aren't easily duplicated for program listings, so the conventions described here will be used instead.

If a letter or numeral (or any character) is underlined, it means the corresponding graphics character is to be used. (A is the spade symbol on the PET.)

Lowercase letters indicate PET special functions:

clr	Clear Screen	hm	Home Cursor
rt	Cursor Right	lft	Cursor Left
up	Cursor Up	dn	Cursor Down
rvs	RVS field on	off	RVS field off
cr	RETURN key	sp	SPACE key

Sp in a line indicates leading or more-than-one blank. For example, dn/sp/sp/HELLO THERE means Cursor Down space HELLO space THERE.

Two IEEE 488 Instruments

The two instruments described here are typical in the way they are controlled via the IEEE 488 bus. Most instruments are controlled by sending and receiving ASCII characters, which are mnemonics of the function being controlled. For example, the HP clock uses the letter D to increment its days' counter. Numbers are usually sent as ASCII strings—in the same way that PRINT provides an ASCII string of digits to a terminal. CR and LF usually indicate a message's end.

Some instruments will use more difficult formats. Two popular forms are BCD, in which two digits per byte are sent, and pure binary, where the value 0-255 is sent. Be sure you know the *exact* formats used by your instruments! Most instruments are unforgiving of bad data; and the responses range from ignoring meaningless characters to the instrument's unaddressing and leaving the bus. Check your instrument's manual!

The HP 59309A Digital Clock

The HP clock is almost the simplest instrument that uses the IEEE 488 bus. Your options are to either set the time or read the time.

When the clock is addressed to talk, it will provide a string of characters with the time in the following format:

(sp or ?) sp NNDDHHMMSS cr lf

The first character is a space or a question mark. If the clock hasn't been set since the last power-off, the question mark will indicate this. The next two digits indicate the month, from 01 to 12. Then comes the day of the month, 01 to 31. (The clock keeps track of the days in each month correctly and has a leap-year switch). Then the hours (00 to 23), minutes and seconds are sent. The carriage return and line feed indicate the end of the message.

Inside the clock are switches that provide variations of the format—colons or commas can either separate the fields, i.e., NN:DD:HH:MM:SS, or simply send the 24-hour time.

When the clock is addressed to listen, eight ASCII characters are used for control:

P—Stop the clock
T—Start the clock
R—Reset the 01:01:00:00:00
S—Each S will increment the Seconds counter
M—Increment Minutes counter
H—Increment Hours counter
D—Increment Days counter
C—Note time, send it when addressed to talk.

For example, the following string will reset the clock to Jan 5, 8:07:12 AM.

PRDDDDHHHHHHHHMMMMMMMSSSSSSSSSSSST

The T at the end restarts the clock.

The HP 8165A Programmable Signal Source

This is a "cadillac" 488 instrument—the front panel of this machine has 41 buttons for selection of modes and a 12-button number pad for entering times, and frequencies. This works out to 35 different command formats for setting up parameters and switch settings and nine commands for telling the controller the machine's setting or starting a sequence of actions. Some of the formats include:

F1—Select Sine Wave
F2—Select Triangle Wave
F3—Select Square Wave
FRQ f MZ—Select frequency in MHz. f is a number from 1 to 9999.
FRQ f MZ—Same for Hz
FRQ f KHZ—Same for kHz
SET:—Report all parameters currently operating when addressed to talk.
SET: n—Report setting in memory # n (0-9)

The 8165 can store up to ten complete settings in its memories, so the SET commands permit the controller to find out what's in the 8165.

An instrument of this complexity is usually programmed with a set of special-purpose programs as needed. Writing a general-purpose BASIC program would be both tedious and wasteful. My experience is that the hardest part is to get the PET and the instrument to communicate. Once that is accomplished, the rest is easy.

work just fine, and so two instruments and the PET can live in harmony together.

A Gotcha

I decided to turn off the 8165 with the PET set up for two instruments as described above. Sure enough, strange things happened.

The clock worked fine:

```
GOTO 10
0130052525
```

And just for fun, look what happens with the 8165 (which isn't on):

```
GOTO 100
F1 D2 I2 FM0 AM0
```

The 8165 has some internal batteries to store and memorize settings until it is turned on again. It also will respond to the IEEE 488 bus.

Now to try things in reverse—the clock doesn't have any batteries. (Clock is off; 8165 is on.)

```
GOTO 100
F1 D2 I2 FM0 AM0      The 8165 is fine
GOTO 10
F1 D2 I2 FM0 AM0      What's this?
```

The 8165 will reply to any address if it is the only device on the bus. The clock acts in the same way. (I don't know if this is a PET fault or an HP design decision. Check your device.)

If your program is intended for more than one device, this can be a disaster. Make sure all required devices are operating when using multiple devices on the bus.

I ran into another gotcha: the 8165 wouldn't accept every frequency change. I tracked this problem down to the presence of the HP clock on the bus. When I turned the clock off, everything worked fine. When debugging, remember to have only one device on your bus.

Common Errors

In theory, if you have understood everything to this point, you can now get an IEEE 488 instrument and make it play with your PET. In practice, this won't happen.

Finding errors is the hardest part of programming, and when you work with the IEEE bus, you can make many mistakes that don't look like errors. When you are able to see errors easily and immediately, you won't need this article.

Here is an incomplete list of the common errors in wait for the unwary IEEE/PET programmer.

The misplaced address. The PET's IEEE addresses are from 4 through 30. The addresses 0 to 3 are reserved for the PET's other I/O devices:

0—Keyboard
1—Tape unit #1
2—Tape unit #2
3—Video screen

If you OPEN a file to the reserved addresses, you won't be speaking to the IEEE bus!

If a device isn't running when the PET wants to talk to it, you will usually get a ?DEVICE NOT PRESENT ERROR. However, if some other device is operating on the bus, you might get the other device's response instead. This happened to me with the HP clock and the 8165. If one was turned off, the other would respond, even though the OPEN statement was referring to the inactive device. This can badly confuse your program.

Time-outs. The PET will only wait for 64 milliseconds before giving up on a device that is slow to respond to the IEEE 488 handshake. Though the IEEE 488 is supposed to work at *any* speed, you may wonder what to do if a device on the bus has failed. If the PET were to wait for a response, there would be no way to return to the user. The 64 ms interval was chosen from the timers available on the 6522 VIA chip, which can count up to 65535 at the 1 MHz clock rate of the PET.

Most instruments will respond within the 64 ms interval, and the PET will read and write the data correctly. This was true of the HP instruments at my disposal. To exercise the PET time-outs, I attached both the clock and the 8165 to the bus, and then OPENed a file to a nonexistent address:

```
NEW

10  INPUT#3,A$
20  IF ST THEN PRINT"ST IS" ST
30  PRINT A$
40  A$=""

OPEN 1,7      (Open the clock to file 1)

OPEN 2,8      (Open the 8165 to file 2)

OPEN 3,10     (The nonexistent device)
```

The little program attempts to input from the nonexisting device. The ST value is a reserved BASIC variable used by the PET for indicating I/O conditions. If ST isn't zero, something went awry.

Now to talk a bit to the devices to wake them up:

```
PRINT #1,"R"      (And the clock resets)
PRINT #2,"E0"     (And the 8165 puts out
                   a signal)
```

If a look at ST is made, all's well:

```
PRINT ST
 0
```

This may take a few tries to work right.

Now to try that nonexistent device:

```
PRINT #3, "HELLO"
```

Looks OK, right? Well, let's see . . .

```
PRINT ST
 -128
```

This is the PET's ST code for "device not present."

Now to try the little program:

```
GOTO 10
ST IS 2

READY.
```

The ST code is 2, which is the time-out for reading data; the nonexistent device didn't say anything. Recall that line 30 said to print A$. The PET *did* print A$, which was an empty string.

The solution to this dilemma is to keep on trying! Write a loop that redoes the INPUT# or PRINT#. In most cases, a slow device will send its characters rapidly enough—once it has its message ready.

Consider these two sample loops:

```
100  PRINT #5," some message or other"
110  IF ST = 1 THEN 100
200  INPUT #6,B$
210  IF ST = 2 THEN 200
```

If you want to mask for certain bits, you can use the AND operator, but parentheses are needed. The above examples would read:

```
110  IF (ST) AND 1 THEN 100      and
210  IF (ST) AND 2 THEN 200
```

The removal of the parentheses makes the PET see the expression as:

```
IF ST AND 1    looks like    IF S TAN D 1
```

which will result in a ?SYNTAX ERROR. Use parentheses or rearrange the order of operations in these cases.

The literal principle. PET outputs to a file the same characters that it sends to the screen. This is also true for the IEEE 488. The PET's format for PRINTing a number is:

(space or – sign) (digits) (optional exponent) (cursor right)

This can raise havoc with an IEEE device that is expecting a character after the number.

Consider the following example:

```
10  PRINT "clr";      (clear screen)
20  FOR J = 1 TO 10
30  PRINT "**********"
40  NEXT J
50  PRINT "hm";       (home cursor)
60  FOR J = 1 TO 10
70  PRINT J"IS A NUMBER"
80  NEXT J

RUN

 1 IS A NUMBER*****
 2 IS A NUMBER*****
 3 IS A NUMBER*****

  etc
```

The asterisk after the number comes from the cursor right character that was sent to the screen. The cursor right follows any numbers sent to the IEEE 488 bus.

The following program sets the frequency of the 8165.

```
10  OPEN 1,8  (The 8165 is at address 8)
20  FOR J = 1000 TO 2000 STEP 10
30  PRINT #1,"FRQ"J"HZ"
40  FOR K = 1 TO 1000
50  NEXT K    (This is a 3 second delay
               loop)
60  NEXT J
```

When this is RUN, the 8165 gives all signs of distress. The frequency appears on the front panel, but the LED that indicates correct entry stays blinking (not completed). Also, the scope shows no change. The PET screen blinks at intervals, indicating that EOI is made true now and then. (I suspect the instrument is making this happen.)

The following modification will fix this:

```
30  PRINT#1,"FRQ"STR$(J)"HZ"
```

The STR$ function converts a number to the string that would be PRINTed, without the cursor right at the end! The general fix for numbers is simple: convert *all* numbers to strings before putting on the IEEE 488 bus.

Fractions. Now that the frequency example is working right, how about trying some other STEP sizes. Here is a simple change:

```
20  FOR J = 1 TO 2 STEP .01
30  PRINT #1,"FRQ"STR$(J)"KHZ"
```

The J loop was changed to do the same thing, but in kilohertz. Line 30 was changed to reflect this. When RUN, it all works fine until about 1.25 kHz—the 8165 now shows 1.259 kHz instead of 1.260. A look at J gives us the clue we need:

```
BREAK IN 40      (Press STOP key)
PRINT J
 1.25999999
```

The PET slips up when computing with fractions . . . and this eventually shows up. The fraction .01 becomes a repeating binary decimal, and after repeated addition, the round-off appears as a slight reduction of the number being added to. In this case, 1.260 turns into 1.25999999.

Catching this is easy . . . if J were put onto the screen first!

```
35  PRINT STR$(J)
```

If you do this, the first "blow up" comes at 1.22999999. Now you are faced with a programming problem: how to get around nasty numbers. One way is to take the INT function, such as:

```
STR$(INT(J*100 + .5)/100)
```

which rounds the number in the hundredths place. More complex tricks will be needed if the PET insists on scientific notation, such as

```
2.35E - 03
```

PRINT your IEEE output onto the screen while debugging.

Next month, we will wrap up our three-part series with a further look at the programming style with the IEEE 488. ■

The PET IEEE 488 File I/O Statements

The PET sees the IEEE 488 bus as a file, and the file I/O statements apply to IEEE 488 transfers. Be sure you know the cassette file I/O before tackling the IEEE 488 bus.

The PET file I/O statements are:

● OPEN (file number), (device number), (secondary address), (filename)

OPEN instructs the PET to associate the file number with the desired I/O device. BASIC uses the file number in its PRINT#, INPUT# and GET# statements to determine where the I/O is to take place. The file number may be from 1 to 255.

The device numbers are assigned as follows:

0 – Keyboard
1 – Cassette unit #1
2 – Cassette unit #2
3 – Screen
4 – 30 IEEE 488 bus

This implies that your IEEE device must be addressed in the range of 4 to 30. Most IEEE devices have a switch or jumpers that permit the changing of their addresses.

The secondary address and filename are optional. However, if you want to use the filename, the secondary address must also be included. The secondary address has the range of 0 to 31.

If the filename is not specified, the OPEN statement sends nothing to the IEEE 488 bus. When BASIC sees the PRINT#, INPUT# and GET# statements, the device number (and secondary address, if specified) are put on the IEEE bus as part of the usual transfer sequences.

If a filename is specified, (i.e., A$ or "SOME NAME"), the OPEN statement activates the IEEE bus making ATN true and sends:

LISTEN (to the appropriate device)
SECONDARY ADDRESS (ORed with 11110000)
FILENAME (all characters)

This permits suitably complex command sequences that require ATN to be true to be sent. If the command sequence has to be repeated later, CLOSE the file and OPEN it again. I haven't been able to check if the above assertions about the filename are true. If you have a bus analyzer, check this out!

● PRINT# (file number), (values to be sent)

First, don't use the abbreviation ?# ; it won't work (when executed, you will see ?SYNTAX ERROR) and will list as PRINT#. Spell out PRINT completely!

The PRINT# sets ATN true and sends the device number as a LISTEN address. If a secondary address as specified, it will be sent also. The device number and secondary address are taken from the appropriate OPEN statement.

ATN is then made false, and the values to be sent are transmitted as ASCII characters in exactly the same way as they would be sent to the screen. For example, if a number is sent, a cursor right character follows the last digit. If you use "," to separate columns, lots of cursor rights are sent. If the PET feels a number should be in scientific format (i.e., 1.53E – 07), that's what is sent! EOI is made true with the last character of data sent.

After the values are sent, an UNLISTEN is sent (with ATN true), and all listening devices are set free.

● INPUT# (file number), (values to be input)

INPUT# sets ATN true and sends the device number as a TALK address. If a secondary address was specified, it will be sent too. The pertinent OPEN statement is used for these values.

ATN is then made false, and the PET accepts characters from the device to the PET's input buffer. If the talker activates EOI, a carriage return is added to the end of the buffer.

After the characters are accepted and carriage return or EOI is recognized, the PET sets ATN true and sends an UNTALK, which releases the device.

BASIC then scans the input buffer in the same way that an ordinary INPUT statement looks at what is typed in. This means that commas and quotes will have the same effects as with normal INPUT. It is best to use an INPUT (string) form and hope your device doesn't send any commas!

As with cassette INPUT#, an 80-character buffer is used. If more than 79 characters arrive without a carriage return, the PET will go into "limbo," and all is lost. (New PETs have this fixed. Over 80 characters are ignored (or worse, the buffer is initialized, and the first 80 characters are lost!). If you have a new PET, try it with cassettes and find out what happens.

INPUT# is susceptible to "time out," and ST should be checked for a time out. Repeat the INPUT# if a time out is detected.

● GET# (file number), (value for entry)

GET# sets ATN true and sends the device number as a TALK address and the secondary address, if specified. ATN is made false, and a single character is accepted.

Then, the UNTALK with ATN true is sent, and the character given to BASIC. For the reasons that make GET X unusable, be sure to only use the GET# (string) form.

The assertion of the UNTALK after GET# makes transmission of multicharacter messages from devices impractical, as most devices will try to repeat their message on repeated application of GET#.

As with INPUT# ST should be checked for a time out, and if timed out, the GET# should be repeated.

● CLOSE (file number)

CLOSE releases the I/O assignments. The PET will allow a maximum of ten files OPEN at one time, and CLOSE will let you reuse an I/O assignment. If you OPEN more than ten files, old PETs will go into limbo and all will be lost. New PETs presumably have this fixed.

If the corresponding OPEN statement had a filename specified, CLOSE sets ATN true and sends the device number and secondary address (ORed 11100000). This feature is intended for PET peripherals.

● CMD (file number), (values to be sent)

CMD initiates the same sequence as PRINT# and sends the values, if any, in the same way that PRINT# does. When finished, CMD does not send the UNLISTEN, so any devices addresses with CMD will listen to further CMDs or PRINT# to the IEEE bus.

All of BASIC's output will be routed to the device defined in the OPEN statement for the file number. If the PET is in command mode, this includes the READY., error messages and LIST. If in run mode, any BASIC printouts, from PRINT to the screen, will go to the IEEE bus instead. A PRINT# will recover from the effects of CMD.

If you are using CMD in command mode, the cursor may not echo the RETURNs you press. The PET will "echo" your keystrokes, but any outputs from BASIC will vanish to the IEEE device. The PRINT# to your IEEE device is the safest recovery from CMD. Remember that any editing of a BASIC program will destroy all variables. This includes open files and CMDs.

● ST (status word)

After each I/O operation, the PET sets the value of a special variable named ST, which will hold its value until the next I/O operation. So the best policy is to check it immediately! The values of ST for the IEEE bus are:

1 Timeout on write
2 Timeout on read (This one should *always* be checked)
64 EOI true
– 128 Device not present

The PET waits for 64 milliseconds to see if a device will respond to the IEEE handshake. If the device doesn't, the I/O operation is quietly aborted, and ST is set. If you are INPUT#ing, you will get "nothing" or zeroes back. If you are PRINT#ing, everything seems to be all right. If your device is slow to respond, checking ST is mandatory.

PRINT#, INPUT# and GET# will return the ?DEVICE NOT PRESENT error if the bus is in an illegal state (which is true if the bus has no devices or the LISTEN or TALK isn't responded to). ST will also be set.

● LOAD, SAVE and VERIFY

The old PETs have a severe error in their IEEE software which prevents the functioning of LOAD, SAVE or VERIFY. The ATN line was left true during the data part of the transfer. This is why owners of old PETs who purchase the PET disk get the new ROMS; the disk won't function with the old ROMs.

The format is the same as with tapes:

LOAD (filename), (device number)
SAVE " "
VERIFY " "

Once the IEEE bus is set to listen or talk, the first four bytes must contain the beginning and ending address + 1 of the block to be transferred. The transfer is then done as pure binary until finished. The bus is then released with an UNT or UNL as needed.

VERIFY will say ?VERIFY ERROR and set ST to 16 if any mismatches were found between the incoming data and the core image in the PET's memory. Since my PET is an old model with the original ROMs, I haven't been able to check LOAD, SAVE and VERIFY for the IEEE 488 bus.

Get Your PET On the IEEE 488 Bus

The final stop on this three-part tour.

Gregory Yob
Box 354
Palo Alto, CA 94302

Commodore's printer and disk use the secondary addresses to control special functions within each device. The secondary address extends the range of allowable addresses on the IEEE 488 bus and is included after the LISTEN or TALK address with ATN made true. Most IEEE devices do not use secondary addresses.

The secondary address permits the device to distinguish between data transfers (for example, file I/O via the disk) and command sequences (for example, to initialize a new disk). The following is a brief summary of the secondary addresses used by Commodore's devices.

PET Printer.

0 – Normal printing. The printer accepts characters and prints them as received.

1 – Formatted printing. The characters are accepted and rearranged according to an internally stored format specification.

2 – Format specification. The characters specifying the format to be used are accepted by the printer.

3 – Pagination control. Accepts a number indicating the number of lines per page.

4 – Control of diagnostic messages. If desired, diagnostic messages will be printed when errors are found. For example, if a number overflows its format, a message indicating this will be printed. This secondary address controls the options to use this feature.

5 – Load programmable character. The printer accepts bytes that specify the dot matrix for one programmable character.

PET Disk.

2 to 14 – Disk "channels" data transfers. The PET disk can have from zero to five files open at once. Each file is defined with an OPEN statement of the form:

OPEN (Log Addr), (Device Addr), (Channel Number), (Command String)

The channel number is a secondary address in the range of 2 to 14. The command string specifies the file type and drive. For example, "0,FILEONE, SEQ, WRITE" means open the file named FILEONE on drive 0 as a sequential file for write only access.

15 – Disk command channel. A variety of commands to the disk is sent via PRINT# to a file opened to the secondary address of 15. The disk can also send error and diagnostic messages to the PET through this channel.

Though it is possible to control complex devices in this manner, these methods can become awkward and clumsy if many data transfers are needed, as is the case for disks and printers. Commodore chose this method to avoid having to modify or extend the PET's BASIC.

Ironically, Commodore now offers a machine-language program, WEDGE, which functions as an extension to BASIC for control of the PET Disk.

Two Examples

In most applications of IEEE instruments, your task will extend beyond communicating with the device. Once communications with the device are established, there remains the conversion of the data to a form usable by people or some other instrument that uses a different form of data. Also, care should be taken to make human communications as pleasant as possible. If your application is in a production (that is, for daily use, and not as an occasional experiment), clarity and reliability are important.

Two BASIC programs, which illustrate how the HP Clock and the HP Signal Source might be used in real-life situations, follow. They are presented here as examples of programming style with the IEEE 488.

Example 1: The HP Clock

Part 1 (*Microcomputing*, July 1980) describes the codes used for the HP Clock with the IEEE 488 bus. Listing 1 interacts with the HP clock in a "human-workable" form. Let's first take a look at how the program is seen from the outside (often called "human engineering" or "the user interface").

When the program is RUN, the following message appears on the screen:

```
HP CLOCK PROGRAM
PRESS ANY KEY WHEN YOU HAVE THE
CLOCK CONNECTED VIA THE IEEE 488
AND THE POWER ON.
```

This reminds the user to connect the clock on the bus and turn on the clock's power. If the PET tries to address a device that isn't connected or turned on, the ?DEVICE NOT PRESENT error message will appear and stop the program. Unfortunately, there is no graceful way to prevent this and keep the program running (some versions of BASIC have error traps; i.e., ON ERROR 5 GOTO . . .).

After you press a key, the request appears:

Listing 1. HP Clock program.

```
10 REM NICE HP CLOCK PROGRAM
20 PRINT"clr HP CLOCK PROGRAM"
30 PRINT"dn dn PRESS ANY KEY WHEN YOU HAVE THE
40 PRINT"CLOCK CONNECTED VIA THE IEEE 488
50 PRINT"AND THE POWER ON.
60 GET A$:IFA$="" THEN 60
70 REM INITIALIZE
80 DIM M$(12),M(12)
90 FOR J=1 TO 12:READ M$(J),M(J):NEXT
100 DATA JAN,31,FEB,28,MAR,31
110 DATA APR,30,MAY,31,JUN,30
120 DATA JUL,31,AUG,31,SEP,30
130 DATA OCT,31,NOV,20,DEC,31
140 INPUT"dn dn CLOCK'S DEVICE ADDRESS:",AD
150 IF AD<3 AND AD>16 THEN 170
160 PRINT"SORRY, LEGAL ADDRESSES ARE 4 - 15":GOTO 140
170 OPEN 1,AD
180 INPUT"dn dn IS THIS A LEAPYEAR";L$
190 IF LEFT$(L$,1)="Y" THEN M(2)=29:PRINT"BE SURE TO SET
    THE CLOCK TO 366 DAYS"
200 REM TIME SETTING REQUEST
210 INPUT"dn dn SET THE TIME";L$
220 IF LEFT$(L$,1)="Y" THEN GOSUB 1000
230 REM DISPLAY TIME
240 GOSUB 2000
250 GOTO 210

1000 REM TIME SETTING ROUTINE
1010 PRINT"clr sp SET THE DATE"
1020 PRINT"dn dn ENTER MONTH AND DAY IN THE FORM:
1030 PRINT"dn sp sp sp sp sp MONTH (SPACE) DAY
1040 PRINT"dn FOR EXAMPLE: sp sp MARCH 25
1050 INPUT"dn";MD$
1100 REM PARSE OUT MONTH & DAY
1110 MI$=LEFT$(MD$,3)
1120 FOR MN=1 TO 12
1130 IF MI$=M$(MN) THEN 1200
1140 NEXT MN: PRINT"dn dn I DON'T RECOGNIZE THE MONTH.
1150 PRINT"PLEASE SPELL THE MONTH COMPLETELY.
1160 PRINT"dn dn PRESS ANY KEY TO TRY AGAIN
1170 GETA$:IFA$="" THEN 1170
1180 GOTO 1010
1200 FOR J=1 TO LEN(MD$)
1210 IF MID$(MD$,J,1)=" sp " THEN 1300
1220 NEXT J
1230 PRINT"dn dn YOU FORGOT THE DAY
1240 GOTO 1160
1300 DY=VAL(MID$(MD$,J))
1310 IF DY>0 AND DY<M(MN)+1 THEN 1400
1320 PRINT"dn dn YOUR DAY IS INCORRECT. IT MUST BE
1330 PRINT"FROM 1 TO"M(MN)"."
1340 GOTO 1160
1400 REM COMPUTE NUMBER OF DAY TICKS
1410 DT=0: IF MN=1 THEN 1430
1420 FOR J=1 TO MN-1: DT=DT+M(J):NEXT J
1430 DT=DT+DY-1
1450 REM DT IS # OF DAYS TO ADVANCE

1500 PRINT"clr sp SET THE TIME"
1505 PRINT"dn dn ENTER THE TIME IN THE FORM:
1510 PRINT"dn sp HOUR : MINUTE : SECOND : AM OR PM
1520 PRINT"dn FOR EXAMPLE: sp sp 2:25:36:PM"
1530 PRINT"dn";:GOSUB 4000
1600 REM PARSE OUT HOURS,MINS,SECS,AM?PM
1610 T$=T$+"XX":TH=VAL(T$)
1620 GOSUB 3000: IF PT>0 THEN 1700
1630 PRINT"dn YOU DIDN'T INCLUDE EVERYTHING
1640 PRINT"PLEASE ENTER ALL FOUR ITEMS WITH
1650 PRINT"COLONS BETWEEN EACH OF THEM
1660 PRINT"dn PRESS ANY KEY TO TRY AGAIN
1670 GETA$:IFA$="" THEN 1670
1680 GOTO 1500
1700 T$=MID$(T$,PT+1)
1710 TM=VAL(T$)
1720 GOSUB 3000: IF PT=0 THEN 1630
1730 T$=MID$(T$,PT+1)
1740 TS=VAL(T$)
1750 GOSUB 3000: IF PT = 0 THEN 1630
1760 T$=MID$(T$,PT+1,2)
1800 REM ERROR MESSAGES
1810 IF TH<1 OR TH>12 THEN PRINT"dn dn YOUR HOURS MUST
     BE FROM 1 TO 12":GOTO 1660
1820 IF TM<0 OR TM>59 THEN PRINT"dn dn YOUR MINUTES
     MUST BE FROM 0 TO 59":GOTO 1660
1830 IF TS<0 OR TS>59 THEN PRINT"dn dn YOUR SECONDS
     MUST BE FROM 0 TO 59":GOTO 1660
1840 IF T$="AM" OR T$="PM" THEN 1860
1850 PRINT"dn dn PLEASE USE AM OR PM ONLY":GOTO 1660
1860 REM AM/PM LOGIC
1870 IF T$="AM" AND TH=12 THEN TH=0
1880 IF T$="PM" AND TH<12 THEN TH=TH+12
1900 REM SET CLOCK AT LAST
1910 PRINT#1,"RP";
1920 IF DT>0 THEN FOR J=1 TO DT:PRINT#1,"D";:NEXT
1930 IF TH>0 THEN FOR J=1 TO TH:PRINT#1,"H";:NEXT
1940 IF TM>0 THEN FOR J=1 TO TM:PRINT#1,"M";:NEXT
1950 IF TS>0 THEN FOR J=1 TO TS:PRINT#1,"S";:NEXT
1960 PRINT#1,"T"
1970 RETURN

2000 REM DISPLAY TIME
2010 PRINT"clr sp sp sp sp sp THE CURRENT TIME IS
2020 PRINT"dn dn sp sp DATE:"
2030 PRINT"dn dn sp sp TIME:"
2040 PRINT"dn dn dn dn dn dn dn dn dn dn dn dn dn dn      (14 dn's)
       sp sp PRESS ANY KEY TO SET TIME
2050 GETA$:IFA$<>"" THEN RETURN
2060 REM FETCH TIME
2070 INPUT #1,T$
2080 IF LEFT$(T$,1)="?" THEN GOSUB 5000
2090 REM PARSE OUT PARTS
2100 T1=VAL(MID$(T$,1,2))
2110 T2=VAL(MID$(T$,3,2))
2120 T3=VAL(MID$(T$,5,2))
2130 T3$=MID$(T$,5,2)
2140 T4$=MID$(T$,7,2)
2150 T5$=MID$(T$,9,2)
2160 PRINT"hm dn dn dn rt rt rt rt rt rt rt rt rt rt"M$(T1);T2
2170 REM AM/PM CALCS
2180 T6$="AM"
2190 IF T3>11 THEN T6$="PM"
2200 IF T3>12 THEN T3=T3-12
2210 IF T3=0 THEN T3=12
2220 T3$=RIGHT$(STR$(T3),2)
2250 PRINT"dn dn rt rt rt rt rt rt rt rt"
     T3$":"T4$":"T5$":"T6$
2500 GOTO 2050

3000 REM SCAN T$ FOR COLONS
3010 FOR PT=1 TO LEN(T$)
3020 IF MID$(T$,PT,1)=":" THEN RETURN
3030 NEXT PT
3040 PT=0: RETURN
4000 REM FETCH STRING VIA GET DUE TO
4010 REM FLAKEY PET INPUT STATEMENT
4015 T$=""
4020 GET A$: IF A$<>"" THEN 4100
4030 PRINT"rvs sp lft";:FOR J=1 TO 300: NEXT
4040 GET A$: IF A$<>"" THEN 4100
4050 PRINT"off sp lft";:FOR J=1 TO 300: NEXT
4060 GOTO 4020
4100 PRINT"off sp lft";
4110 IF A$=CHR$(13) THEN PRINT: RETURN
4120 PRINT A$;: T$=T$+A$: GOTO 4020

5000 PRINT"hm dn dn dn dn dn dn dn dn dn dn dn dn
     >>>>>>> sp TIME NEEDS TO BE SET sp<<<<<<<"
5010 PRINT"dn>>>>>>> sp DUE TO POWER FAILURE sp <<<<<<<"
5020 T$=MID$(T$,3):RETURN
```

CLOCK'S DEVICE ADDRESS;?

Now enter the address on the DIP switches for the device. If an unacceptable value, such as 16, is entered, the PET will respond with:

SORRY, LEGAL ADDRESSES ARE 4-15

and ask again. The best way to avoid problems is to forbid illegal values for inputs, tell the user that he has goofed and mention the correct range of values.

Once the device address is in, the PET asks:

IS THIS A LEAP YEAR?

If "YES" is entered, a reminder appears to set the clock accordingly.

BE SURE TO SET THE CLOCK TO 366 DAYS

The last request asks:

SET THE TIME?

If the user doesn't want to set the time, the screen clears and the date and time are shown:

THE CURRENT TIME IS
DATE: JAN 29
TIME: 7:02:54 PM

PRESS ANY KEY TO SET TIME

The time ticks away with the seconds changing the most rapidly. A different set of values will appear on the clock:

01 29 19 02 54

The program has translated from 24-hour time to normal AM/PM time and changed the month from a number to the month's name.

The HP clock will send a ? as the first time character if the clock has not been set since a loss of power. If you pull the plug on the clock and plug it in again, the program will stop with a ?DEVICE NOT PRESENT ERROR. When the program is RUN, the time will be displayed with the following in the space

between the time and the PRESS ANY KEY line:

```
>>>>>TIME NEEDS TO BE SET<<<<<
>>>>>DUE TO POWER FAILURE<<<<<
```

Now if you press a key, the SET THE TIME? request will reappear:

```
SET THE TIME? YES
```

The screen clears and will display:

```
SET THE DATE
ENTER MONTH AND DAY IN THE FORM:
    MONTH (SPACE) DAY
FOR EXAMPLE: MARCH 25
? JANUARY 29
```

If the first three letters in the month are incorrect, the program will make you start over:

```
I DON'T RECOGNIZE THE MONTH.
PLEASE SPELL THE MONTH COMPLETELY.
PRESS ANY KEY TO TRY AGAIN.
```

If you missed the date, the PET says:

```
YOU FORGOT THE DAY
PRESS ANY KEY TO TRY AGAIN
```

If you enter an inappropriate date, such as JAN 45, the PET, will say:

```
YOUR DAY IS INCORRECT. IT MUST BE FROM 1 TO 31.
```

The program has the number of days for each month stored inside. If the month were February, the range 1 to 28 would have been shown instead.

Now that the date is entered correctly, the screen clears to let the time be entered.

```
SET THE TIME
ENTER TIME IN THE FORM:
HOUR : MINUTE : SECOND : AM OR PM
FOR EXAMPLE: 2:25:36:PM
7:19:25:PM (you enter this line)
```

The screen will flicker a bit, and then the time display will appear.

The PET won't correctly input a string with colons in it, so the entry here is "faked" to look like a normal INPUT line. Unfortunately, if you must INST or DEL to correct your line, the correction won't really be entered. This can be programmed around, but I didn't feel like doing it with an instrument on loan to me for a week. The subject of faking INPUT is an article in itself.

Again, there are some error messages to help and assist the user:

```
YOU DIDN'T INCLUDE EVERYTHING
PLEASE ENTER ALL FOUR ITEMS WITH
COLONS BETWEEN EACH OF THEM
PRESS ANY KEY TO TRY AGAIN
YOUR HOURS MUST BE FROM 1 TO 12
YOUR MINUTES MUST BE FROM 0 TO 59
YOUR SECONDS MUST BE FROM 0 TO 59
PLEASE USE AM OR PM ONLY
```

Here, a bad entry only forces you to reenter the time. The date is OK, so why redo it?

Perhaps this example is extreme. In many situations it isn't worth the programming time to make a program completely convenient to use. As an idealist, I wrote it up to show what can be done if ease of use is required.

HP Clock BASIC Program Review (Listing 1)

Lines 10 to 60 announce the program and force the user to wait until he has made sure the HP Clock is attached to the PET's IEEE 488 and the power is turned on. DATA in lines 100 to 130 are placed in the months' names' array M$ and the months' lengths' array M.

Lines 140 to 170 request the HP Clock's address and check to see if the address is legal. Line 160 tells the user to try again and mentions the legal range as a hint. Lines 180 and 190 take care of the leap-year problem by changing the month length for February to 29 days and reminds the user to check the leap-year switch on the HP Clock.

In lines 200-220, the user is asked if the time is to be set (which must be done when the clock is first used), and a loop is entered in lines 240 and 250. Subroutine 1000 sets the time, and subroutine 2000 displays the time. The program will not leave subroutine 2000 until a key is pressed. Line 250 jumps to the time-change request as needed.

Setting the time in subroutine 1000 is a complicated job, requiring correctly entering the data. First, you must enter the month and day as explained in lines 1010 to 1040, which give an example of the expected format.

Line 1050 picks up the user's entry, and lines 1000 to 1180 take a look at the first three characters to see if they fit a month's name. Lines 1140 to 1180 take care of any mistake in the entry of a month's name.

Lines 1200 to 1220 scan the input string, MD$, until a space is found. This removes the remnants of the month's name and brings us up to the date digits. Failure to find a space means the day was forgotten, and the user is told to start all over.

Lines 1300 to 1340 check the day number with the number of days in the month M(MN). If everything is OK, lines 1400 to 1450 will figure out the value DT, which is used to send the correct number of Ds to the clock for date setting.

Now that we have the number of days from Jan. 1 (in the number DT), lines 1500 to 1530 will tell the user to enter the time in a familiar format—HH:MM:SS:AM or PM. Subroutine 4000 is used to enter the string T$ via the GET statement. In lines 1620 to 1850, the string T$ is snipped apart at the colons, and each part is examined for the correct range of values; subroutine 3000 looks for the colons, and lines 1680 to 1760 do the scissor-work. We eventually end up with the values TH, TM, TS and T$, for hours, minutes, seconds and AM/PM values.

Lines 1860 to 1880 adjust the hours, TH, according to the AM or PM value. Lines 1900 to 1970 set the HP Clock—first the clock is reset via "RP," and then the correct numbers of "D," "H," "M" and "S" are sent to set the time. Then "T" is sent to start the clock.

Subroutine 2000 sets up the screen in lines 2010 to 2060. Note that the GET in line 2050 only checks if a character was entered. If not, it will continue to line 2070. The HP Clock is accessed in line 2070, and line 2080 checks for "?." The "?" means the clock saw a power failure, and subroutine 5000 will warn of this event.

Lines 2100 to 2150 get the various parts of the HP Clock's message. T1 is the month number; T2 is the day number. Line 2160 displays the month and day values.

Lines 2170 to 2220 adjust the hours value, T3$, to reflect whether an AM or PM time is being shown. Then line 2250 prints the hours, minutes, seconds and AM/PM marker.

In subroutine 3000, PT is the position of the first colon found in the string T$.

Subroutine 4000 simulates a cursor and constructs T$ from the characters entered through GET A$. No editing is provided, so if you make an error, the entry must be repeated. A little more code could catch A$ = 20 (code for DEL) and give some limited editing (equivalent to back space or rubout on a terminal).

Subroutine 5000 puts the power failure message on the screen and strips the "?" from T$. This permits the display of time code to work correctly.

The astute programmer will note that no provision is made for bad messages from the HP clock (which might make the program fail in some cases). You should check the values T1, T2, T3, T3$, T4$ and T5$ for their legal values and make another attempt to read the time made in case of an error. In the event of several consecutive errors, the program should mention this to the user.

There are limits to how "fail-safe" a program must be made. In many cases, malfunctions will not be critical, and it isn't worth the effort required to make the program survive the errors. I do not recommend the PET for any real-time control applications that may result in injury or loss of property in the event of the PET's failure!

Example 2: The HP 8165A Signal Source

Part 1 introduced the 8165A. Naturally, your interest will be with the devices that you have available, and the example shown here is a "laboratory application"; that is, a program similar to one you might want to build for your instrument.

Let's pretend that the response of a stereo amplifier needs to be tested in a production line. The frequencies and voltages to be tested are:

```
10 Hz,   Sine Wave,     1.000 volts
10 Hz,   Square Wave,   1.000 volts
20 Hz,   .....
20 Hz,   .....
50 Hz,
```

Test sine wave and square wave responses at 1.000 volts for 10, 20, 50, 100 . . . up to 20 kHz.

The plan for a program is as follows:

1) Initialize. For example, open

```
10 PRINT"clr STEREO TEST PROGRAM
20 PRINT"dn dn BE SURE THE 8165 IS ON AND THAT
30 PRINT" THE IEEE 488 IS CONNECTED.
40 PRINT"dn REMEMBER THE ADDRESS FOR THE 8165
50 PRINT"MUST BE 8. PLEASE CHECK THIS.
60 GOSUB 1000
70 OPEN 1,8
80 REM SET UP 8165
90 PRINT#1,"FRQ1ØHZAMP1.ØØØVF1D2OD"
100 REM HOOK UP STEREO
110 PRINT"clr STEREO AMPLIFIER TEST"
120 PRINT"dn ATTACH THE NEW UNIT TO THE
130 PRINT"TEST STATION."
140 GOSUB 1000
200 REM PERFORM TEST
210 PRINT"clr >>>> TEST IN PROGRESS<<<< "
220 FOR L1=1 TO 4
230 FA=1Ø↑L1
240 FOR L2 = 1 TO 3
250 IF L2 = 1 THEN FR=FA/1ØØØ
260 IF L2 = 2 THEN FR=FA*2/1ØØØ
270 IF L2 = 3 THEN FR=FA*5/1ØØØ
275 IF FR >25 THEN 430
280 FOR W = 1 TO 2
290 IF W=1 THEN W$ = "SINE"
300 IF W=2 THEN W$ = "SQUARE"
310 REM SET 8165 UP
320 PRINT#1,"FRQ"STR$(FR)"KHZ"
330 IF W=1 THEN PRINT#1,"F1OE"          (letter F, numeral 1,
340 IF W=2 THEN PRINT#1,"F3OE"           letters OE)
350 REM SET TIMER & REPORT
360 T1 = TI                             (tee one = tee eye)
370 PRINT"hm dn dn dn TEST AT:";
380 PRINT"sp sp FREQ:"FR*1ØØØ"sp sp"W$"sp sp sp"
390 IF TI - T1< 6ØØ THEN 390
400 REM TURN 8165 OFF
410 PRINT#1,"OD"                         (letters OD)
420 NEXT W
430 NEXT L2
440 NEXT L1
450 REM TEST COMPLETE
460 PRINT"clr ****** TEST COMPLETED ******"
470 PRINT"dn dn REMOVE AMPLIFIER FROM TEST STATION"
480 GOSUB 1000
490 GOTO 110

1000 PRINT"dn dn PRESS ANY KEY WHEN READY"
1010 GET A$:IF A$="" THEN 1010
1020 RETURN
```

Listing 2. Stereo Test program.

the IEEE 488 file.

2) Tell the operator to hook up an amplifier

3) Start the test

4) Loop through the frequencies for each frequency

5) Loop through sine and square

6) Wait for 10 seconds before continuing

7) Report where the test is on the screen

8) End of both loops

9) Tell the operator the test is finished

10) Go to step 2

Listing 2 shows these steps in a BASIC program. From the user's point of view, when the program is RUN, the message below appears:

STEREO TEST PROGRAM

BE SURE THE 8165 IS ON AND THAT THE IEEE 488 IS CONNECTED.

REMEMBER THE ADDRESS FOR THE 8165 MUST BE 8. PLEASE CHECK THIS.

PRESS ANY KEY WHEN READY

This reminder ensures that the 8165 is properly connected, powered and addressed. The PET program won't work if these conditions aren't met.

Now it is time to test a unit. The screen clears (after a key is pressed) and displays:

STEREO AMPLIFIER TEST

ATTACH THE NEW UNIT TO THE TEST STATION.

PRESS ANY KEY WHEN READY

Now the test commences, with a report on the current frequency and waveform being used:

>>>>>TEST IN PROGRESS<<<<<

TEST AT: FREQ: 200 SQUARE (current freq & waveform)

After about two minutes (each frequency and waveform takes ten seconds), the screen clears and tells the user:

•••••TEST COMPLETED•••••

REMOVE AMPLIFIER FROM TEST STATION

PRESS ANY KEY WHEN READY

Now we are ready to perform another test. Look at the scope and notice that the output of the 8165 is turned off between tests and between mounting the new amplifiers. Though unimportant in this example, more serious equipment should always be set to a "safe" state when the operator has to handle the equipment.

Lines 10 to 60 in the BASIC code state the program's name and remind the user to check the address setting on the HP 8165. Subroutine 1000 waits for you to press a key.

Three nested loops are used to scan through the frequencies and waveforms. The L1 loop sets the frequency decade from the range 10-99 Hz to 10000-99999 Hz. The L2 loop is used to select between 1, 2 and 5 times the frequency selected by L1. W chooses between sine and square waves.

Lines 200 to 300 compute the frequency FR in two steps (FA is set to 10^{L1}, and FR is set to 1,2 or 5 times FA), and W$ is set to report sine or square. In line 275 the top value to be tested is 20000 Hz, so to terminate the loops requires a test of the frequency larger than 20000 Hz.

Instead of using 20000 for the test, I am using 25000. (If you look at the code, FA is in kilohertz, so the test is for 25.) Due to the PET's way of computing numbers, when L1 is 3 and L2 is 2, FA turns out to be a tiny amount over 20, which terminates the test too soon.

When testing for equality or differences, make sure the number in the PET is what you think it is. Most floating point numbers will be slightly (and unprintably) different than the value you want, so fudge accordingly.

Line 320 sends the correct command to the 8165 for frequency. Note that FR is sent as the string STR$(FR). This avoids the Cursor Right after the number, which could totally confuse the 8165. Lines 330 and 340 specify the waveshape by directly sending the correct set of characters to the 8165. "OE" turns the 8165 on.

Lines 350 to 390 print the test values and wait for 600 jiffies, or ten seconds. When they are finished, line 410 turns the 8165 off (this is a "safe" state; e.g., during hook-up, the test leads could be shorted).

Lines 450 to 490 announce the end of the test and tell the user to remove the stereo amplifier. Note that the 8165 is in the "off" state.

I will leave it to you to figure out how to use the HP clock to control the timing of the stereo test program (Listing 2, part 2) instead of the PET's internal clock. Another variation is to put up the time each test is run for logging purposes.

More "Gotchas"

Program bugs. When I was debugging the HP Clock program (see Listing 1), the days' count wouldn't come out right. Some months tended to have two or three too many days, while others ran short. For example, May 5 put May 11 on the clock, and February 10 showed February 7.

I first thought that the IEEE 488 device was miscounting characters. I checked by printing the number sent onto the screen. The error wasn't here.

The eventual source of the problem was that the routine that counted the total days in

Function	Old Pet (hex)	(dec)	New PET (hex)	(dec)
Send TALK (MTA)	F0B6	61622	F0B6	61622
Send LISTEN (MLA)	F0BA	61626	F0BA	61626
Send UNTALK	F17A	61818	F17F	61823
Send UNLISTEN	F17E	61822	F183	61827
Set ATN true and send character in accumulator	F0BC	61628	F0BC	61628
Send data character in accumulator••	F0F1	61681	F0EE	61678
Get data character in accumulator	F187	61831	F18C	61836
Flag byte	0222	545	00A5	165

••Set flag byte to FF (255) before calling this routine.

Table 1. PET IEEE ROM and RAM locations.

the previous months just added the same number each time. For May, it added 31 four times, and for February, it added 28 once!

Another bug came from the "hidden bits" in PET numbers. In the Stereo Test program (Listing 2), there was the following line:

```
IF FR>20 THEN....
```

The testing program stopped at 10 kHz instead of 20 kHz. When I printed FR, I got 20. FR was formed from the two computations:

```
FA = 10↑L1
FR = FA*2/1000
```

The PET's exponentiation operator isn't totally exact, so a few bits slipped through. The division didn't help, and FR ended up a slight amount over 20, which is enough to make the condition true. The cure was to test for more than 25 instead.

These errors are subtle. If you aren't a total expert on your PET, these are nearly impossible to find.

```
10 REM PET SERIAL OUTPUT
20 REM GREGORY YOB
30 PT = 826
40 READ BT: IF BT  0 THEN 60
50 POKE PT,BT: PT=PT+1: GOTO 40
60 DIM BD(6),RT(6)
70 FOR J=1 TO 6
80 READ BD(J),RT(J)
90 NEXT J
100 PRINT"clr SERIAL OUTPUT"
110 PRINT"dn PARITY"
120 PRINT"0=EVEN, 1=ODD, 2=MARK"
130 INPUT P
140 IF P=0 THEN 180
150 IF P=1 THEN 180
160 IF P=2 THEN P=255: GOTO 180
170 GOTO 110
180 POKE 994,P
190 PRINT"dn BAUD RATE"
200 INPUT BT
210 FOR J=1 TO 6
220 IF BT=BD(J) THEN 380
230 NEXT J
240 PRINT"RATES ARE:"
250 FOR J=1 TO 6. PRINT BD(J): NEXT
260 GOTO 190
380 POKE 995, RT(J)
390 PRINT"# TIMES TO REPEAT CHAR"
400 INPUT N
410 N=INT(N): IF N<Ø OR N>255 THEN 390
420 PRINT"PRESS ANY KEY TO SEND CHARS"
430 GET A$: IF A$+"" THEN 430
440 PRINT A$
450 POKE 997,N: POKE 992, ASC(A$)
460 SYS(826)
470 GOTO 420

1000 DATA 173,4,2,234,234,240,1
1010 DATA 96,173,64,232,41,64,240
1020 DATA 241,120,21,192,3,144,2
1030 DATA 88,96,32,98,3,32,153
1040 DATA 3,88,76,58,3,234,24
1050 DATA 173,224,3,96,234,169,0
1060 DATA 141,225,3,173,224,3,162
1070 DATA 1,160,0,24,74,144,5
1080 DATA 160,225,238,225,3,72,152
1090 DATA 157,240,3,104,232,224,8
1100 DATA 208,234,273,226,3,48,12
1110 DATA 240,3,238,225,3,173,225
1120 DATA 3,41,1,240,2,169,255
1130 DATA 157,240,3,96,162,255,232
1140 DATA 189,240,3,141,34,232,172
1150 DATA 227,3,173,0,64,173,0
1160 DATA 64,173,0,64,136,208,244
1170 DATA 234,236,228,3,208,228,96
1180 DATA 96,0,0,0,0,0,0
1190 DATA 0,24,173,229,3,208,2
1200 DATA 56,96,173,224,3,206,229
1210 DATA 3,96,0,0,0,0,0
1220 DATA 0,0,0,0,0,0,0
1230 DATA 0,0,0,0,0,65,2
1240 DATA 0,195,11,0,0,0,0
1250 DATA 0,0,0,0,0,0,0
1260 DATA 0,255,0,0,0,0,0
1270 DATA 255,0,255,255,0,0,0
1280 DATA 0,0
1300 DATA -1
1999 REM PARAMETERS FOR BAUD RATES
2000 DATA 9600,5,4800,11,2400,23
2010 DATA 1200,48,600,97,300,195
```

Listing 3. Serial output via the IEEE 488 bus port.

Using the PET ROM

Since the PET knows the IEEE bus, there have to be routines in the PET ROM that know how to work the bus. A year ago, some of my clients' requirements forced me to access the PET's ROM for the IEEE. (One had a machine that didn't like the PET's state between IEEE messages; the other wanted to know the PET's maximum IEEE transfer rate.)

Table 1 indicates the pertinent RAM and ROM locations for the PET IEEE routines. Use caution when working with these, as I have only been able to check the ones mentioned below. In one case, a routine sent a character at an apparent rate of 5000 characters/second! (The listener didn't see anything at all.) The routine in question took a look at the bus, decided the bus wasn't in a legal state and returned, instead of sending the character! If you have an accurate PET disassembly, here is a good place to use it.

Input from the IEEE Bus. This can be approached either from machine language or as a mix of machine language and BASIC. In all cases, the first step is to open a file to the bus via BASIC. (This must be done; make sure that only one file is open.)

The next step is to send a TALK to the device. From BASIC, this is a SYS(61622), and in machine language it is a JSR F0B6 (or 20 B6 F0).

To handshake a character in requires calling the machine language in ROM. Here's a catch: the character arrives in the A register. From BASIC, you must SYS to a short routine that performs JSR F187 and an STA (somewhere) (and RTS to get back). Then PEEK (somewhere) gets your character. The machine code in hexadecimal is 20 87 F1 8D xx xx 60. The xx xx is your "somewhere." The value from the IEEE bus is the complement of your character; that is, the 1's and 0's are exchanged.

Send to the IEEE Bus. Again, the first step is to open a file to the bus and be sure that only one file is open. Then, send the ATN LISTEN via SYS(61626). (In machine language, JSR F0BA, or 20 BA F0.) Now, put the complemented value into location $0222 with a POKE 546, CHARACTER.

The last step is to SYS (61681), which sends the character. In some cases, you will have to set a flag first by setting location $021D to $FF by POKE 541,255. I have used both methods with success.

The machine-language sequence is A9 FF 8D 1D 02 20 xx xx 8D 22 02 20 F1 F0 60. The 20 xx xx is a JSR to your routine at xx xx, which gets a character in the A register.

Both the input and the output will leave the device active on the bus. Make ATN true and send the UNL and UNT value to release the device.

The IEEE lines in the PET don't have to be used for the IEEE 488 bus. There are 12 easily used bits of parallel I/O that can be controlled with suitable PEEKs and POKEs, and two PET Hard Copy Easy," *Kilobaud Microcomputing*, September 1979, p. 100.

Printing Hazards

The difference between the PET's display and character codes and the ASCII character set causes some difficulties when you use the IEEE 488 bus for printed output.

1. ASCII printers use the most significant bit (MSB) as a parity bit. If the PET is sending a graphics character (or lowercase, as provided by the POKE 59468,14 for old PETs), the printer will either ignore this and print the corresponding ASCII for the seven less significant bits or print a "parity error" character. If you get a parity error character, set your printer to the "no parity," or "mark" parity, option.

2. The PET cursor control characters result in the ASCII values in the range 0 to 31, which are control characters in ASCII. If you are lucky, these will be ignored; if you aren't, some of these may result in setting your printer to unwanted modes. (The Comprint printer is

Listing 4. Serial output, machine-language assembly listing.

```
     This code was hand assembled and then patched - so the flow
isn't continuious and there are occasional NOPs that aren't needed.

                          ! Check SHIFT key
033A AD 04 02  SENSE      LDA SHIFT (0203)    read shift key location
     EA EA                NOP, NOP            (tis a patch)
     F0 01                BEQ GO (0342)
     60                   RTS                 back to'BASIC if SHIFT
                                               pressed
                          ! See if device is ready
0342 AD 40 E8  GO         LDA $E840           Get all PB2 lines from VIA
     29 40                AND #40             Mask NRFD bit
     F0 F1                BEQ SENSE (033A)    Wait if not ready

                          ! Set up PET for transmission of characters
                          !  Turn off interrupts
                          !  Get character
                          !   Set carry if no more characters
                          !  Set up Xmission table
                          !  Send character

0349 78                   SEI                 Interrupts off
034A 20 C0 03             JSR FETCH (03C0)    Fetch Character
                                               (Set up as a subroutine
                                                to let you "roll your own"
                                                routine)
     90 02                BCC GO1 (0351)
     58                   CLI                 Interrupts on. If Carry is
     60                   RTS                 set, no more chars to send.
                                               If you make your own FETCH,
                                               use this convention.

0351 20 62 03  GO1        JSR SETUP           Set up Xmit table for char in A
     20 49 03             JSR XMIT            Send char
     58                   CLI                 restore interrupts
     4C 3A 03             JMP SENSE           Look at SHIFT key again
     EA                   NOP                 (patch)

035C 18        FFETCH     CLC                 Dummy version of FETCH
     AD E0 03             LDA CHAR (03E0)     Test Char location
     60                   RTS
     EA                   NOP                 (guess)

                          ! Set up Xmission Table
0362 A9 00     SETUP      LDA #00
     8D E1 03             STA PARITY (03E1)   Initialize parity counter
     AD E0 03             LDA CHAR (03E0)     Get char
     A2 01                LDX #01             X reg counts/7 bits of char.
                          ! Shift char & If carry set, load FF into
                          ! Xmit table. If carry not set, load 00
                          ! (NOTE: Start & Stop bits are assumed preset
                          ! in Xmit table. Be sure this is so in your
                          ! program too.)

036C A0 00     SLOAD      LDY #00             Y holds 00 or FF for bit
     18                   CLC                   in char.
     4A                   LSR A               Shift LSB into Carry
     90 05                BCC HOPPITY         Bit is zero
     EE E1 03             INC PARITY (03E1)   '1' bit adds to parity count
     48                   PHA                 Stuff A on stack
     98                   TYA                 Y to A
     9D F0 03             STA START,X         Put into Xmit table. I just
                                               love non-symmetrical
                                               instruction sets! (6502
                                               has no Y indexed addressing)
     68                   PLA                 Restore A from stack
     E8                   INX                 On to next bit
     E0 08                CPX #08             7 bits yet?
     D0 EA                BNE SLOAD (036C)    no, repeat

                          ! According to  option, set the parity
                          ! bit in the Xmit table

0382 AD E2 03             LDA POPTION (03E2)  Get option value
     30 0C                BMI MARK            MSB means MARK parity
     F0 03                BEQ EVEN            zero is EVEN
     EE E1 03             INC PARITY          Add 1 for odd parity
     AD E1 03  EVEN       LDA PARITY
     29 01                AND #01             LSB has parity in it
     F0 02                BEQ ZILCH           Save LDA #00 if A is 00
     A9 FF     MARK       LDA #FF
     9D F0 03  ZILCH      STA START,X         Put in Xmit table. X happens
     60                   RTS                 to be right value!

                          ! Send Character
0399 A2 FF     XMIT       LDX #FF             The next instruction
     E8        CONT       INX                   makes X zero.
     BD F0 03             LDA START,X         Get byte to send
     8D 22 E8             STA $E822           Put on IEEE DIO Lines (out)

                          ! Delay loop for baud rate
03A2 AC E3 03             LDY RATE (03E3)     Get countdown value
03A5 AD 00 40  AGAIN      LDA $0400           4 cycles of delay
     AD 00 40             LDA $0400             ditto
03AB AD 00 40             LDA $0400             ditto
```

a "lucky" one.)

3. As a result of these first two steps, if you use CMD and LIST, the listings you get will have missing or misleading characters. I have a program (drop me a card) that will list a BASIC program in a legible form.

4. The PET does not transmit a line feed. You must provide CHR$(10) after every carriage return.

5. If your printer needs a carriage return delay, either print the required number of CHR$(0) or insert a small waiting loop; i.e., FORJ = 1TO20:NEXT.

6. Most printers have no formatting capabilities. If you keep careful count of the number of characters sent, formatting is clumsy, but possible. Pitfalls include:

- A printed number has a CHR$(29) sent after the last digit, which is not a space and is usually ignored by printers.
- TAB and SPC provide CHR$(29), and not spaces. (New PETs have this fixed.)
- LEN(STR$(number)) will not count a CHR$(29), since STR$ produces a string without a blank or skip after the last digit.
- If the number is small or large, beware of scientific format; i.e., 1.23E+23.

7. If you are attempting a word-processing program, the PET's codes for the lowercase characters and the ASCII codes are different. The PET thinks the lowercase letters lie in the range 192 to 223, and ASCII likes the range 96 to 127.

A further complication is that the ASCII character set and the PET character sets don't match. Backarrow on the PET is ASCII underline; the curly brackets, vertical bar and tilde in ASCII don't exist on the PET. The ASCII accent mark (looks like a reverse apostrophe) is seen by the PET as a space. Your printer might have other character options to puzzle you.

Wrapping It Up

Working with the IEEE 488 bus is nearly an entire engineering discipline in itself. I hope my efforts enable you to get

```
   88                DEY                 reduce countdown
   D0 E4             BNE AGAIN (03A5)     keep going till count is zero
   EA                NOP                 Successful branch takes 3
                                          so this compensates to
                                          make a 17 cycle per loop
                                          delay

   EC E4 03          CPX BITCOUNT        Check number of bits to
                                          be sent.
   D0 E4             BNE CONT            Do next bit
   60                RTS

.......(some room here) .......

                     ! Fetch Character for real. Feel free to
                     ! make your own routine. Set carry bit when
                     ! out of characters.
03C0 18      FETCH   CLC                 Be sure to do this!
     AD E5 03        LDA CHCOUNT (03E5)  # chars to send
     D0 02           BNE OK
     38              SEC                 Set  carry, out of chars
     60              RTS

     AD E0 03  OK    LDA CHAR            Get char - you might use
                                          TAX & LDA CHAR,X here.
     CE E5 03        DEC CHCOUNT         decmt chars counter
     60              RTS

....... (some room here) .......

                     ! Data Area

03E0 00      CHAR    ! Character to send. (Move elsewhere if you
                         want to send more than one)
03E1 00      PARITY  ! Parity Counter
03E2 00      POPTION ! Parity Option. 0-even,1-odd,FF-mark
03E3 00      RATE    ! Initial countdown for baud rate. POKEd
                       by the BASIC program.
03E4 00      BITCOUNT ! Number of bits to send (10 or 11 decimal)
03E5 00      CHCOUNT ! Number of chars to send

....... (a gap again) ......

03F0 00      START   ! Start of Xmit table
03F1 00 00 00 00 00 00 00    ! Character, lsb first
03F8 00              ! Parity bit
03F9 FF FF           ! Stop bit(s)
```

aboard the IEEE 488 bus of your PET and turn it to some profitable use. ■

References

1. "IEEE Standard Digital Interface for Programmable Instrumentation," IEEE Std 488-1975, ANSI MC 1.1-1975.

2. Hewlett-Packard, 1502 Page Mill Road, Palo Alto, CA or PO Box 301, Loveland, CO 80537. Several publications are available on request.

3. "PET 2001-8 User's Manual" and "PET Communication with the Outside World," Commodore Business Machines.

4. "Getting Aboard the 488-1975 Bus," Motorola.

5. "PET User Notes," PO Box 371, Montgomeryville, PA 18936.

6. MOS Technology, Inc., 950 Rittenhouse Road, Norristown, PA 19401.