DEPARTMENTS

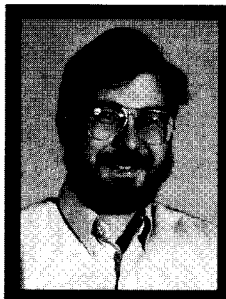
How the PC Keyboard Got its Bits

FIRMWARE FURNACE

Ed Nisley

The only thing new in the world is the history you don't know.
Harry Truman

History is more or less bunk.
Henry Ford



Ed takes a brief aside from protected-mode programming to look at the keyboard. After reviewing the history of keyboard development, Ed gets into the nitty-gritty issues of keyboard microprocessors and scan codes.

Truman is a better guide than Ford when you deal with the innards of PC hardware. As we've seen many times in this column, even the most bizarre PC designs made perfect sense at the time. Yes, with years of hindsight, we can all do a better job. But, I doubt I would have done as well back then.

The IBM PC's keyboard facilities are a case in point. The closer you look, the more bizarre they become. Three separate processors, a bidirectional serial interface, read- and write-only ports, mysterious status and control bits, overlapping key codes, and a wealth of rarely used options all cry, "Kludge!" at the top of the data sheets.

After I collected all my notes for this topic, I realized that the subject is incomprehensible without more background than fits in a single column. This month, I'll explain how the keyboard got into its present condition and make Demo Taskette 3 in the FFTS kernel display the default system scan codes. Next month, we'll delve into unexplored territory.

So, let us begin...

IN THE BEGINNING

Contrary to what many people think, the Original IBM PC wasn't the

first product the IBM Corporation produced. The PC designers ran a classic Skunk Works operation, drawing whatever they could use from the rest of the company and ignoring whatever didn't fit. As a result, the PC contained some things old, some things new, some things borrowed, and yes, some things Blue.

The PC keyboard's pedigree, in particular, stretches back to the legendary IBM Selectric and the infamous O-series keypunches. Among other advances, IBM honed capacitive-sensing key technology to a fine edge; the bold tactile click you either loved or hated was designed in, not added on. The only catch was the manufacturing cost-if you ever see an old PC keyboard, pry off a keycap and admire the small parts!

In any event, the keyboard had three main sections: typewriter keys in the middle, 10 function keys to the left, and a block of keys to the right that served for both cursor control and numeric data entry. Each key had an identification number, assigned more or less left to right and top to bottom within its group.

An Intel 8048 microcontroller scanned the capacitive key matrix and sent a scan code to the PC when each key was pressed. Believe it or not, each key's scan code exactly matched its key number. The Esc key in the upper-left corner of the main typewriter area was Key 1 and returned scan code 01. Key numbers are in decimal and scan codes are in hex just to keep you on your toes.

Each time a key went down [a *key make* in IBM parlance), the 8048 stored its scan code in a 16-entry buffer. All keys had typematic action. In other words, if the key remained down, the 8048 stored an additional make code after about 500 ms and another code every 100 ms after that. When the key went up (a *key break),* the 8048 set the high bit of the key's scan code and stored it in the buffer. The Esc key's break code was 81 hex.

Figure 1 shows the overall scheme. Makes a lot of sense so far, doesn't it?

The keyboard transmitted scan codes from the buffer to the PC

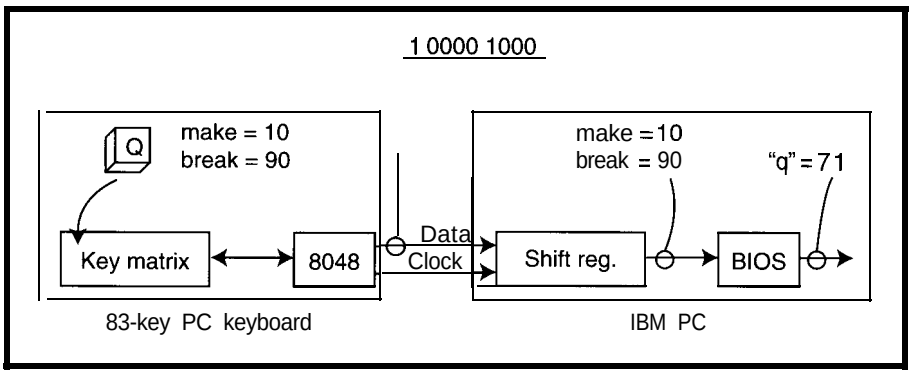


Figure 1--The Original IBM *PC 83-key* keyboard *scan* codes *were simply the key numbers, assigned left to right and fop to bottom. The break code for each key was equal to the scan code with bit 7 sef high. A high start bit preceded each code through the serial interface and triggered an interrupt when* if emerged from *the* shiff *register inside the PC. The BIOS translated the scan codes info a standard set of key codes that depended on the shift keys.*

through a four-wire cable: +5 V, ground, Data, and Clock. A fifth line, -Keyboard Reset, wasn't used and vanished from later specs. Clock and Data were open-collector TTL lines, nominally bidirectional, but in point of fact, the PC didn't have much to say to the keyboard.

The PC's keyboard interface was charmingly simple: an 8-bit shift register driven by the keyboard's Clock and Data lines. The 8048 sent a "1" start bit before clocking eight scan code bits into the shift register. When the start bit emerged from the shift register, it set a latch that triggered IRQ 1 and pulled the keyboard Clock line low to prevent further transmissions. After reading the shift register, the PC toggled an I/O port pin that cleared the shift register, reset the interrupt, and enabled the keyboard's Clock line in preparation for the next scan code.

Unlike many keyboards of the time, the PC keyboard had n-key rollover. You could, for instance, hold down the Ctrl, Alt, and Del keys at once and be sure the BIOS would receive each scan code in the correct order with typematic action intact. Most key chords didn't make much sense, but a TSR cottage industry arose around the ability to detect finger exercises such as left Shift, right Shift, Ctrl-S.

The PC's BIOS keyboard functions, unlike those for the serial port and printer, were complete enough that nearly all programs used them. Indeed, to this day many programmers are blissfully unaware of how the BIOS goes about its task. We embedded folks don't have that luxury. In the Protected Land, you can see all the way down to bare silicon and copper!

The BIOS tracked the make and break codes for each shift key to figure out what to do with all the other keys. For example, the BIOS translated Key 16 (make code 10 and break code 90) into character "q" (71 hex], "Q" (51), or Ctrl-Q (11) as appropriate. The Read Keyboard Input BIOS function returned the translated character in the AL register along with the key's scan code in AH to uniquely distinguish the key. The notation 10/71,10/51, and 10/11 seen in the references should make more sense now.

The PC keyboard also included an Alt shift key. The BIOS translation for Alt-shifted keys returned zero in AL and the key scan code in AH: 10/00 for Alt-Q. These keystrokes were usually interpreted as special commands rather than characters, but that was up to the program. Alt applied to several other noncharacter keys and the BIOS suppressed the remainder.

Function, cursor control, and special keys behaved much like Alt-shifted keys. The BIOS returned zero in AL with AH containing the raw scan code, a modified scan code unique to the key combination, or a duplicate of another key's character. The 7-Home key (Key 71, scan code 47), returned 47/00,47/37, and 77/00 for normal, Caps, and Ctrl shifts. Alt-Home was suppressed. The key codings show that Caps-Home is ASCII character "7" and Ctrl-Home is entirely unique.

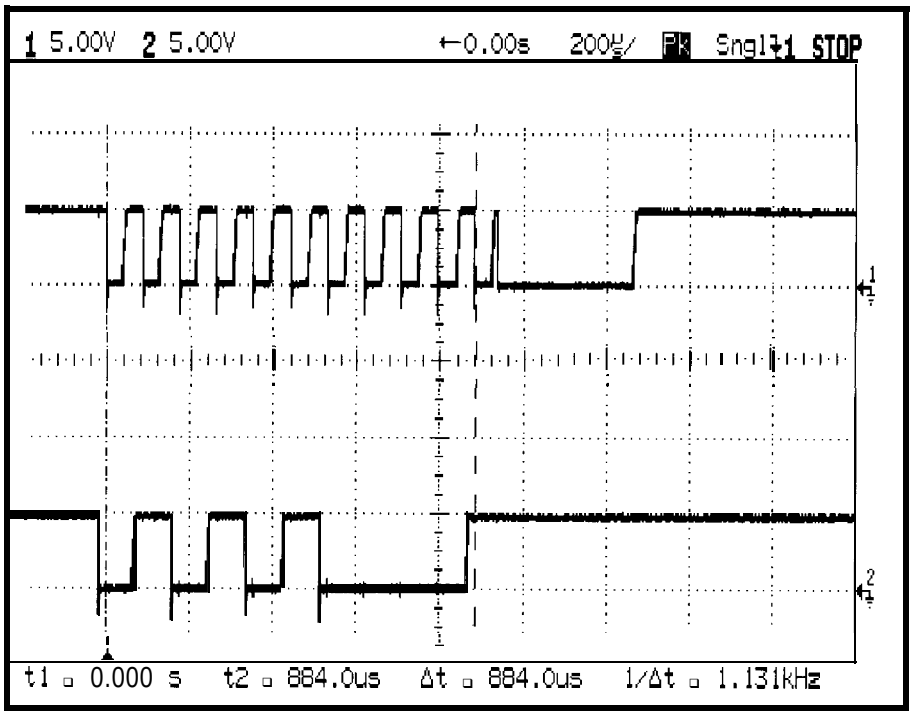


Photo 1—*This scope shot shows the bits flowing through the keyboard cable after pressing the Q key on a 101-key Enhanced keyboard. The falling edges of the clock waveform in Trace 1 mark valid data bits in Trace 2. The start bit, identified by fhe left cursor, is always a low level, followed by fhe eight data bits for make code 15 hex, an odd parity bit, and a high stop bit, marked by the right cursor. The 8042 system keyboard controller inside the PC pulls the Clock line low after it receives the stop bit to prevent any further transmissions until the key is processed.*

Although there are some quirks, on the whole, the scheme makes a lot of sense. Using simple hardware, mapping various shift states into ASCII characters, providing an Alt shift case to expand the codes, and making everything reasonably easy was, ahem, the PC's key to success.

The PC Compatibility Barnacles began growing seconds after the Original PC hit the store shelves.

CONVERTED CODES

The IBM PC/AT (unofficially: "Advanced Technology") ushered in the '286 CPU, protected mode with 64-KB segments, and the all-new, ergonomically advanced, 84-key keyboard.

The 84th key sported a Sys Req label and was supposed to trigger special operating-system functions. PC-DOS (the only operating system that mattered) had no such functions. Application programs ignored it, hard-core utilities put it to a variety of peculiar tasks, and most folks never noticed it.

The Original PC's 83-key keyboard featured sculptured key caps. Although they looked nice, their main function was ensuring that you pressed each key directly over its switch mechanism because the keys jammed if you hit them anywhere else. PC/AT keys used a different mechanical design that stabilized the keys and allowed big, flat key caps. It also had a *slightly* different layout.

The entire numeric keypad slid half an inch to the right and captured the Esc key in its upper-left corner. The open single quotation mark and tilde (`~) key moved from beside the left-Shift key to snuggle in the crook of an L-shaped, aircraft-carrier-sized Enter key. Caps Lock and Ctrl swapped places, much to the relief of Selectric owners and the disgust of everybody else. Another cottage industry sprang up delivering TSRs to unswap the offending keys.

Under the covers, though, every-*thing* changed as IBM renumbered the keys to match their new locations. This made some sense, even though there were some peculiarities. The PC function keys bore labels F1 through F10 in left-to-right, top-to-bottom order, but had key numbers 65-74 in top-to-bottom, right-to-left sequence. The Esc key, formerly Key 1, became Key 90 in its new location, leaving a suspicious 15-key gap after F9. PC pundits wondered what IBM was up to-surely, those missing key numbers meant something.

As the PC and later the AT became a roaring success, IBM began "converging" their myriad disparate keyboard lines into a single offering. The PC keyboard's method of generating break codes by setting bit 7 of the make code limited the number of distinct keys to 128, and IBM realized that a mere 128 keys might not suffice for some mainframe or minicomputer keyboards. Thus, the break codes on the PC/AT's keyboard became two-byte sequences: FO followed by the key make code.

For example, the Q key, formerly Key 16, became Key 17 with make code 15 and break code FO 15. Just to hammer the point home, Sys Req was Key 105 with make code 84 (note the high-order bit!) and break code FO 84. Perforce, the new keyboard was entirely incompatible with the old PC.

Changing the PC's barnacle-encrusted scan codes would shatter every existing PC program. The PC/AT got around that problem by translating the new keyboard scan codes into the familiar system scan codes everyone knew and loved. As far as application programmers were concerned, the new AT keyboard was identical with the old PC keyboard,

Acronyms

CPL	Current Privilege Level
DPL	Descriptor Privilege Level
EOI	End Of Interrupt (command)
FDB	Firmware Development Board
FFTS	Firmware Furnace Task Switcher
GDT	Global Descriptor Table
GDTR	GDT Register
IDT	Interrupt Descriptor Table
IF	Interrupt Flag
IMR	Interrupt Mask Register
IOPL	I/O Privilege Level
LDT	Local Descriptor Table
LDTR	LDT Register
NT	Nested Task
P bit	Present bit (in a PM descriptor)
RF	Resume Flag
RPL	Requestor Privilege Level
TF	Trap Flag
TR	Task Register
TSS	Task State Segment

although the new distinction between keyboard and system scan codes remains obscure to this day.

Scan code translation, however, could not occur at the BIOS level because some ill-behaved programs burrowed through the BIOS directly to the hardware interface. IBM solved that problem by interposing yet another microcontroller between the BIOS and the keyboard cable. Once you have a microcontroller on board, a lot of other things are possible.. .

PASSING THE BUCK

The old PC keyboard sent a start bit followed by eight data bits. The start bit was a 1 that triggered IRQ 1 when it popped out of the system board's shift register. The new system keyboard controller, an Intel 8042, had enough intelligence to support an entirely different keyboard serial protocol.

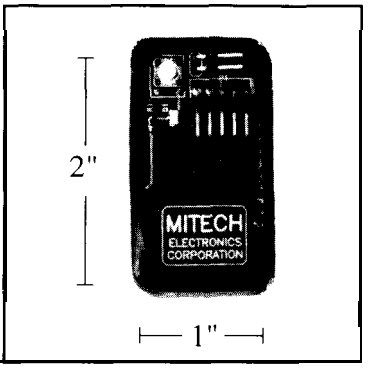
The PC/AT keyboard used a data format much like ordinary RS-232: one start bit (a zero), eight data bits, an odd parity bit, and one stop bit. Unlike RS-232 asynchronous data, a clock signal indicated when each data bit was valid. Photo 1 shows this interface in action. The enhanced keyboard that I'll discuss shortly uses the same signaling technique. The two microcontrollers verify parity and request retransmissions as needed, eliminating most system error handling.

In addition to an 84th key, the PC/AT keyboard also sprouted three indicator LEDs: Caps Lock, Numeric Lock, and Scroll Lock. Nobody in the PC world knew what to do with the latter, but the former two eliminated a cottage industry of on-screen "video LED" indicators. For the first time, the system had something to say to the keyboard, as the three LEDs are controlled by the BIOS rather than directly by the keyboard.

The AT's designers also gave the new system keyboard controller life-and-death responsibility: one of the 8042's output pins drove the system's Reset circuit. As I described in INK 38, resetting the entire system was the only way to get the 80286 CPU out of protected mode. Thus, was born the Worst Hack in PC-dom.

OUR SMALLEST EPROM EMULATOR

Eliminate the need to burn and ***learn***
all in a package about the size of a 9 volt battery



- Super small (about 2"x1"x1")
- Uses Surface Mount Design
- Fits in EPROM socket
- Downloads in less than 7 seconds
- Emulates 2764, 27128, 27256, 275 12
- Access time < 1 OONS
- Plug and Play: Plug cable into computer and download. Once power is supplied to circuit, cable can be removed
- Includes software for IBM compatable PC's
- Loads Intel Hex and Binary files
- Includes Serial Cable & Battery Backup
- Nearly same footprint as EPROM

MITECH
ELECTRONICS
CORPORATION

411 Washington Street,
Otsego, Michigan 49078

TEL: 616-694-4920 FAX: 616-692-2651

Since 1985

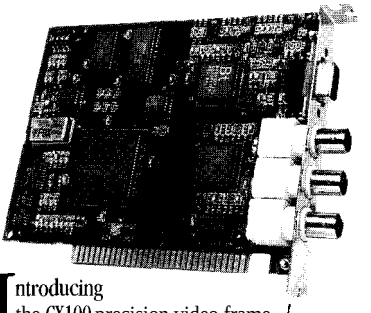
CALL NO Wfor Your Special Introductory Price of $194.95

plus $5.00 shipping and handling

VISA& MASTERCARD ACCEPTED

2

PRECISION FRAME GRABBER FOR ONLY $495*



Introducing the CX100 precision video frame grabber for OEM, industrial and scientific applications. With sampling jitter of only ±3nS and video noise less than one LSB, ImageNation breaks new ground in imaging price/performance. The CXlOO is a rugged, low power, ISA board featuring rock solid, crystal controlled timing and all digital video synchronization. A Software developers will appreciate the simple software interface, extensive C library and clear documentation. The CXlOO is a software compatible, drop-in replacement for our very popular Cortex I frame grabber. A Call today for complete specifications and volume pricing.

ImageNation Corporation
Vision ***Requires Imagination***
800-366-9131

— CXlOO FEATURES —

- Crystal Controlled Image Accuracy
- Memory Mapped, Dual-Ported Video RAM
- Programmable Offset and Gain
- Input, Output and Overlay LUTs
- Resolution of 5 12x486 or Four Images of 256x243 (CCIR 512x512 & 256x256)
- Monochrome, 8 Bit, Real Tie Frame Grabs
- Graphics Overlay on Live or Still Images**
- External Trigger Input
- RGB or B&W, 30 Hz Interlaced Display
- NTSC/PAL Auto Detect, Auto Switch
- VCR and Resettable Camera Compatible
- Power Down Capability
- BNC or RCA Connectors
- Built-In Software Protection**
- 63 Function C Library with Source Code
- Text & Graphic Library with Source Code
- Windows DLL, Examples and Utilities
- Software also available free on our BBS
- Image File Formats: GIF, TIFF, BMP, PIC, PCX,TGA and WPG

** THESE OPTIONS AVAILABLE AT EXTRA COST.
* $495 IS DOMESTIC. OEM SINGLE UNIT PRICE

P.O. BOX 276 BEAVERTON, OR 97075 USA PHONE (503) 641.7408 FAX (503) 643-2458BBS (503) 626.7763

23

In any event, the 8042 translated the new keyboard make and break codes into the new, yet at the same time old, system scan codes. The new Q key returned system make code 10 and break code 90, just like the old Q. The BIOS translation was essentially unchanged because the system scan codes were identical after the 8042 got through with them. Figure 2 shows the successive translations from keycap to BIOS.

About two years after the AT's introduction, IBM dropped another plank: the 101-key Enhanced keyboard. If adding the 84th key was a challenge, imagine seventeen more!

ENHANCED AND ENLARGED

The Enhanced keyboard was known internally as the *Converged* keyboard because it offered all the keys and functions required by IBM's mainframe, minicomputer, and PC divisions. Early users quickly dubbed it the *Concatenated* keyboard in honor of its size and committee ancestry.

The numeric pad drifted another two inches to the right and spat out the Esc key. The empty space refilled with ten dedicated cursor movement, screen control, and editing keys numbered from Key 75 through 89 with a few gaps. Ten function keys fissioned to twelve, migrated en masse to a straight row above the typewriter keys, and became Keys 112 through 123. The Esc key, now Key 110, returned to the far upper-left corner of the keyboard.

PC users wondered why they needed two each of the Ctrl and Alt keys, particularly as the left Ctrl key obstinately remained where Caps Lock should be. Mainframe and minicomputer users, faced with different keycap legends, rejoiced as the Enter key returned to its proper spot below right Shift and worried themselves not at all over broken symmetry.

Moving the function keys gave the keyboard cottage industry a shot in the arm. Giant clone keyboards heaved into view with ten function keys to the west and another dozen up north. Some sported an additional dozen function keys to match the minicomputer layout. Small rodents skittering about the desktop harassed these behemoths, many succumbed to terminal coffee and soda ingestion, and users with no remaining desktop space dealt a death blow.

[The current crop of twisted and bloated ergonomic keyboards are roughly the same size. One wonders if desks have become any larger.. .}

As with the PC/AT's 84-key keyboard, the new Enhanced keyboard was incompatible with all previous PC systems. The Compatibility Barnacles once again dictated that existing programs must work without changes, while new programs should detect and use the new keys.

The new keys between the typewriter keyboard and the numeric pad were not, strictly speaking, new. The numeric pad on 83- and 84-key keyboards served for both cursor movement and numeric entry. For example, pressing the 8-↑ key produced "8" (38 hex) in Num Lock mode and moved the cursor upward in unshifted mode. Heavy-duty spreadsheet users developed finger cramps from shifting and unshifting their way from cell to cell.

The enhanced keyboard separated those functions: the new ↑ key moved the cursor upward, while 8-↑ performed both functions. With the keyboard in Num Lock mode, the numeric pad really was a numeric pad without requiring any additional shift keystrokes.

Now, the question was: how can the BIOS map two keys into the same value while keeping them distinct? The solution is a simple matter of firmware in three different processors....

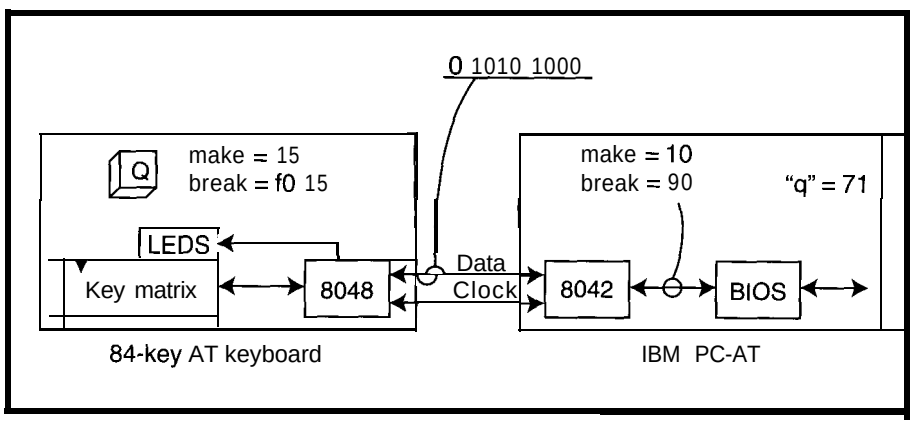


Figure 2—*The IBM* PC/A J *84-key* keyboard used a *completely* different key-numbering *system. The* scan codes were *essentially arbitrary and used all eight data bits. Break codes became a two-byfe sequence: F0 hex followed by the key's scan code. The serial data format used a zero start bit, eight data bits, an odd* parity *bit, and a high stop bit, much like ordinary RS-232 serial communication. An 8042 microcontroller inside the PC/A J handled the new format and translated new scan codes info fhe o/d key codes.*

SHIFTING SANDS

Obviously, the microcontroller in the keyboard (by now an 805 l or one of its ilk) must know which key is pressed. It could return the same code for two distinct keys, but that would suppress valuable information. IBM's keyboard designers decided to break new ground: the keyboard would keep track of the current shift state and send different scan codes for the same key.

Keypad 8-↑ is Key 96 on both the 84-key and Enhanced keyboards. The controller sends make code 75 and break code FO 75 for that key regardless of which shift keys are pressed. That much remained unchanged.

The new ↑ atop the "inverted T" of cursor keys is Key 83. In normal, unshifted mode, it returns make code EO 75 and break code EO FO 75. Even though there are fewer than 128 keys, many of the new keys return scan codes with an EO prefix byte. In this case, the base scan code is the same as for Key 96, but that is not a general rule.

Now it gets weird.

The ↑ make code returns EO FO 12 EO 75 when it's pressed after the left

Listing 1—*This interrupt handler reads system scan codes from the 8042 keyboard controller and places them in a ring buffer without analyzing them at all. A matching routine in Demo Task 3 extracts them for display on the VGA. The 8042 converts the torrent of keyboard scan codes into the slightly more familiar system scan codes. The real-mode BIOS processes system scan codes into characters, but those routines are not available in protected mode.*

```
CODESEG

PROC    KeyHandler
USES    EAX,EDX,DS

MOV     EDX,SYNC_ADDR       ; show a tick
IN      AL,DX
OR      AL,40h
OUT     DX,AL

MOV     EAX,GDT_DATA        ; get addressability to our data
MOV     DS,AX

IN      AL,KEY_DATA         ; read scan code from controller
Punt
CMP     [RingCount],RING_SIZE   ; room for one more?
JE      @@Done              ; nope, bail out

MOVZX   EAX,AL              ; clear high bytes
MOV     EDX,[RingHead]      ; aim at head entry
MOV     [EDX*4 + KeyRingl,EAX   ; save the scan code
INC     [RingCount]         ; account for it

INC     EDX                 ; tick and wrap index
CMP     EDX,RING_SIZE
JB      @@NoWrap
XOR     EDX,EDX
@@NoWrap
MOV     [RingHead],EDX

@Done:
MOV     AL,NS_EOI           ; now reset the 8259 ISRs
OUT     I8259A,AL           ; EOI primary controller

MOV     EDX,SYNC_ADDR       ; remove the tick
IN      AL,DX
AND     AL,NOT 40h
OUT     DX,AL

POP     DS                  ; restore bystanders
POP     EDX
POP     EAX

IRET                        ; return to interrupted code

ENDP    KeyHandler
```

Shift (Key 44, scan code 12) is down. The first byte, EO, introduces an extended scan code. The second byte, FO, signifies a break code. The third byte is the same scan code as Key 44, the left-Shift key, but the EO prefix told you it's not really that key. The last two bytes are the unshifted ↑ make code.

When the right Shift (Key 57, scan code 59) is down, you get EO FO 59 EO 75. The keyboard sends an ersatz break key for the shift key and then the usual make code.

The ↑ break code becomes EO FO 75 EO 12 when the left Shift is down. The controller sends the normal ↑ break code (EO FO 75) and then reshifts with EO 12. The right-Shifted break code is EO FO 75 EO 59, of course.

We're not done yet!

The system keyboard controller translates these keyboard scan codes into somewhat less formidable system

new from
DON LANCASTER

ACTIVE FILTER COOKBOOK
The sixteenth (!) printing of Don's bible on analog op-amp lowpass, bandpass. and highpass active filters. De-mystified instant designs. $28.50

CMOS AND TTL COOKBOOKS
Millions of copies in print worldwide. THE twa books for digital integrated circuit fundamentals About as hands-on as you can get. $24.50 each.

INCREDIBLE SECRET MONEY MACHINE II
Updated 2nd edition of Don's classic on setting up your own technical or craft venture. $18.50

LANCASTER CLASSICS LIBRARY
Don's best early stuff at a bargain price. Includes the CMOS Cookbook, The TTL Cookbook, Active Filter Cookbook, PostScript video, Case Against Patents, Incredible Secret Money Machine II, and Hardware Hacker II reprints. $119.50

LOTS OF OTHER GOODIES

Ask the Guru I or II or III	$24.50
Hardware Hacker II or III	$24.50
Micro Cookbook I	$19.50
PostScript Beginner Stuff.	$29.50
PostScript Show and Tell	$29.50
Intro to PostScript Video	$29.50
PostScript Reference II	$31.50
PostScript Tutorial/Cookbook	$19.50
PostScript by Example	$31.50
Understanding PS Programming	$29.50
PostScript: A Visual Approach.	$22.50
PostScript Program Design	$ 24.50
Thinking in PostScript	$ 22.50
LaserWriter Reference	$19.50
Type 1 Font Format.	$15.50
Acrobat Reference.	$24.50
Whole works (all PostScript)	$380.00
PostScript Insider Secrets	FREE
Hacking Insider Secrets	FREE

POSTSCRIPT SECRETS
A Book/Disk combination crammed full of free fonts, insider resources. utilities, publications, workarounds, fontgrabbing more. For most any PostScript printer. Mac or PC format $29.50

BOOK-ON-DEMAND PUB KIT
Ongoing details on Book-on-demand publishing, a new method of producing books only when and as ordered. Reprints, sources. samples. $39.50

THE CASE AGAINST PATENTS
For most, individuals, patents are virtually certain to result in a net loss of sanity, energy. time, and money. This two volume set shows you tested and proven real-world alternatives. $28.50

BLATANT OPPORTUNIST I
The reprints from all Don's Midnight Engineering columns. Includes a broad range of real world proven coverage on small scale technical startup ventures. Stuff you can use right now. $24.50

RESOURCE BIN I
A complete collection of all Don's Nuts & Volts columns to date, including a new index and hi6 master names and numbers list $24.50

FREE SAMPLES
Well. nearly free anyway. Almost Do join us on GEnie PSRT to sample all of the Guru 6 godies. The downloading cost on a typical Guru is is 21 cents. Modem access: (800)638-8369, then a JOINGENIE. Use DMD524 for your keycode.

FREE VOICE HELPLINE VISA/MC

SYNERGETICS
Box 809-CC
Thatcher, AZ 85552
(520) 428-4073

scan codes. Even though the new make codes can be two bytes long, the 8042 converts them to single-byte values, possibly with an EO prefix for the new keys. The unshifted ↑ make code becomes EO 48 and the break code is EO C8. With the left Shift down, you get EO AA EO 48 and EO C8 EO 2A. You can probably guess that the system scan code for left Shift is 2A. Similarly, with the right Shift down, you get EO B6 EO 48 and EO C8 EO 36.

If you understand that, what happens if both shift keys are down when you press ↑? 'Obviously, the make code is EO AA EO B6 EO 48 and the break code is EO C8 EO 36 EO 2A. What could be easier?

OK, final question: what does left-Shift-PrtSc produce? Would you believe EO 2A EO 37 EO B7 EO AA? That's in addition to the 2A and AA produced by the left Shift key, of course. The PrtSc key is a make-only key, which means you won't get a break code when it goes up. That explains, at least a little bit, why the make code includes the break code.

The BBS files this month display the system keyboard-controller output for each key. The IRQ 1 interrupt handler shown in Listing 1 reads the system scan codes from the controller and places them in a ring buffer. A display routine running in Demo Task 3 extracts the codes and writes them on the VGA display. If you have a scope or logic analyzer, you can also watch the keyboard scan codes on the cable and compare them with the system scan codes.

You should also compare the system scan codes with those in your references. So far, every table I've seen has errors, omissions, or misinterpretations. Some are simple typos (watch those Dees, Ohs, Zeros, Eyes, Ells, and Ones!), while others are more serious. Spend a while tapping the keys, reading the books, and making marginal notes...

Don't forget that there is yet *another* translation layer. All those old application programs called the original BIOS keyboard functions and should not be offended by unexpected key codes. When the BIOS reads a "new" system scan code from the

Listing 2—*This table gives the real-mode BIOS return values for each of the Enhanced keyboard's 101 keys. The comments show the key's Scan Code Set 3 number in hex, the physical key number in decimal, and the keycap legend. Scan codes 13 and 53 correspond to keys found only on non-U.S. 102-key keyboards. Those keyboards lack the key that produces scan code 5C. The keyboard interface described next month will use this table to generate BIOS-compatible values.*

```
            STRUC       KEYCODES
BaseCase    DW          0                       ; unshifted character
CapCase     DW          0                       ; Caps shift
CtrlCase    DW          0                       ; Ctrl shift
AltCase     DW          0                       ; Alt shift
            ENDS        KEYCODES

            LABEL       KeyCodes BYTE
              --- Standard PC codes ----   Sc Key  Keycap
             Base Caps Ctrl Alt           ; Cd Num Legend
KEYCODES <>                               ; 00
KEYCODES <>                               ; 01
KEYCODES <>                               ; 02
KEYCODES <>                               ; 03
KEYCODES <>                               ; 04
KEYCODES <>                               ; 05
KEYCODES <>                               ; 06
KEYCODES <03B00h,05400h,05E00h,06800h>    ; 07112 F1
KEYCODES <0011Bh,0011Bh,0011Bh,00100h>    ; 08110 Escape
KEYCODES <>                               ; 09
KEYCODES <>                               ; 0A
KEYCODES <>                               ; 0B
KEYCODES <>                               ; 0C
KEYCODES <00F09h,00F00h,09400h,0A500h>    ; 0D 16 Tab
KEYCODES <02960h,0297Eh,00000h,02900h>    ; 0E 1 `~
KEYCODES <03C00h,05500h,05F00h,06900h>    ; 0F113 F2
KEYCODES <>                               ; 10
KEYCODES <00000h,00000h,00000h,00000h>    ; 11 58 Left Ctrl
KEYCODES <00000h,00000h,00000h,00000h>    ; 12 44 Left Shift
KEYCODES <>                               ; 13 45 102-key only
KEYCODES <00000h,00000h,00000h,00000h>    ; 14 30 Caps Lock
KEYCODES <01071h,01051h,01011h,01000h>    ; 15 17 Q
KEYCODES <00231h,00221h,00000h,07800h>    ; 16 2 1!
KEYCODES <03D00h,05600h,06000h,06A00h>    ; 17114 F3
KEYCODES <>                               ; 18
KEYCODES <00000h,00000h,00000h,00000h>    ; 19 60 Left Alt
KEYCODES <02C7Ah,02C5Ah,02C1Ah,02C00h>    ; 1A 46 Z
KEYCODES <01F73h,01F53h,01F13h,01F00h>    ; 1B 32 S
KEYCODES <01E61h,01E41h,01E01h,01E00h>    ; 1C 31 A
KEYCODES <01177h,01157h,01117h,01100h>    ; 1D 18 W
KEYCODES <00332h,00340h,00300h,07900h>    ; 1E 3 2@
KEYCODES <03E00h,05700h,06100h,06B00h>    ; 1F 115 F4
KEY CODES <>                              ; 20
KEYCODES <02E63h,02E43h,02E03h,02E00h>    ; 21 48 C
KEYCODES <02D78h,02D58h,02D18h,02D00h>    ; 22 47 X
KEYCODES <02064h,02044h,02004h,02000h>    ; 23 33 D
KEYCODES <01265h,01245h,01205h,01200h>    ; 24 19 E
KEYCODES <00534h,00524h,00000h,07B00h>    ; 25 5 4$
KEYCODES <00433h,00423h,00000h,07A00h>    ; 26 4 3#
KEYCODES <03F00h,05800h,06200h,06C00h>    ; 27116 FS
KEYCODES <>                               ; 28
KEYCODES <03920h,03920h,03920h,03920h>    ; 29 61 Space
KEYCODES <02F76h,02F56h,02F16h,02F00h>    ; 2A 49 V
KEYCODES <02166h,02146h,02106h,02100h>    ; 2B 34 F
KEYCODES <01474h,01454h,01414h,01400h>    ; 2C 21 T
KEYCODES <01372h,01352h,01312h,01300h>    ; 2D 20 R
KEYCODES <00635h,00625h,00000h,07C00h>    ; 2E 6 5%
KEYCODES <04000h,05900h,06300h,06D00h>    ; 2F 117 F6
KEYCODES <>                               ; 30
KEYCODES <0316Eh,0314Eh,0310Eh,03100h>    ; 31 51 N
KEYCODES <03062h,03042h,03002h,03000h>    ; 32 50 B
KEYCODES <02368h,02348h,02308h,02300h>    ; 33 36 H
KEYCODES <02267h,02247h,02207h,02200h>    ; 34 35 G
```

(continued)

Listing 2-continued

```
KEYCODES <01579h,01559h,01519h,01500h> ; 35  22  Y
KEYCODES <00736h,0075Eh,0071Eh,07D00h> ; 36   7  6^
KEYCODES <04100h,05A00h,06400h,06E00h> ; 37 118  F7
KEYCODES <>                            ; 38
KEYCODES <00000h,00000h,00000h,00000h> ; 39  62  Right Alt
KEYCODES <0326Dh,0324Dh,0320Dh,03200h> ; 3A  52  M
KEYCODES <0246Ah,0244Ah,0240Ah,02400h> ; 3B  37  J
KEYCODES <01675h,01655h,01615h,01600h> ; 3C  23  U
KEYCODES <00837h,00826h,00000h,07E00h> ; 3D   8  7&
KEYCODES <00938h,0092Ah,00000h,07F00h> ; 3E   9  8*
KEYCODES <04200h,05B00h,06500h,06F00h> ; 3F 119  F8
KEYCODES <>                            ; 40
KEYCODES <0332Ch,0333Ch,00000h,03300h> ; 41  53  ,<
KEYCODES <0256Bh,0254Bh,0250Bh,02500h> ; 42  38  K
KEYCODES <01769h,01749h,01709h,01700h> ; 43  24  I
KEYCODES <0186Fh,0184Fh,0180Fh,01800h> ; 44  25  O
KEYCODES <00B30h,00B29h,00000h,08100h> ; 45  11  0)
KEYCODES <00A39h,00A28h,00000h,08000h> ; 46  10  9(
KEYCODES <04300h,05C00h,06600h,07000h> ; 47120  F9
KEYCODES <>                            ; 48
KEYCODES <0342Eh,0342Eh,00000h,03400h> ; 49  54  .>
KEYCODES <0352Fh,0353Fh,00000h,03500h> ; 4A  55  /?
KEYCODES <0266Ch,0264Ch,0260Ch,02600h> ; 4B  39  L
KEYCODES <0273Bh,0273Ah,00000h,02700h> ; 4C  40  ;:
KEYCODES <01970h,01950h,01910h,01900h> ; 4D  26  P
KEYCODES <00C2Dh,00C5Fh,00C1Fh,08200h> ; 4E  12  -_
KEYCODES <04400h,05D00h,06700h,07100h> ; 4F 121  F10
KEYCODES <>                            ; 50
KEYCODES <>                            ; 51
```

(continued)

8042, it returns the best match among the "old" codes. For example, unshifted ↑ returns the old 8-↑ AH/AL values: 48/00. Keys that didn't exist on the old keyboard and have no valid equivalents are simply discarded.

New programs, however, can call a different BIOS entry point to get the new extended codes: unshifted ↑ returns 48/E0, while 8-↑ returns the familiar 48/00. This is the level nearly all PC programmers work at-with the mysteries of keyboard and system scan codes carefully hidden from view.

In protected mode, we can't take advantage of the BIOS functions. Given what you know now, are you up to writing a BIOS keyboard replacement?

Me, neither.

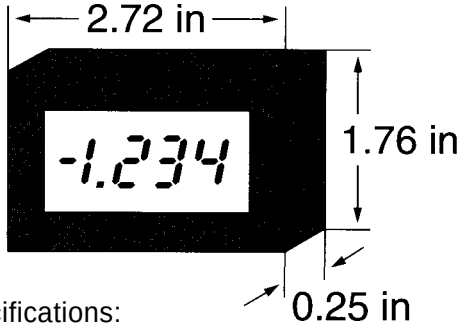
SCANNING THE CODES

Here's where a bit of history comes in handy. Recall that IBM designed the Enhanced keyboard for all its systems. While the convoluted scan codes might be necessary for backwards PC compatibility, they made

3½-DIGIT LCD PANEL METER

-Available now at an unheard of price of $15 plus s&h

New! Not surplus!



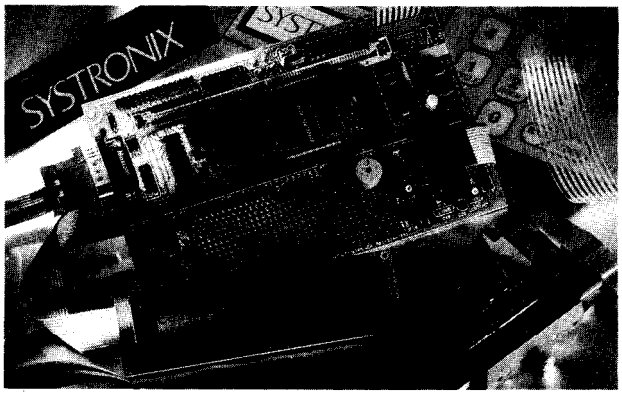
Specifications:

Maximum input: k199.9 mV
additional ranges provided through external resistor dividers
Display: 3½-digit LCD, 0.5 in. figure height, jumper-selectable decimal point
Conversion: Dual slope conversion, 2-3 readings per sec.
Input Impedance: >100M ohm
Power: 9-12 VDC @ 1 mA DC

Circuit Cellar, Inc.

4 Park Street, Suite 12, Vernon, CT 06066
Tel: (203) 875-2751 Fax: (203) 872-2204

NEW! UNIVERSAL DALLAS DEVELOPMENT SYSTEM from $199!



- It's a complete 8051-family single board computer!
- One board accommodates any 40 DIP DS5000, 40 SIMM DS2250, 40 SIMM DS2252, or 72 SIMM DS2251, 8051 superset processor! Snap one out, snap another in.
- Programs via PC serial port. Program lock & encrypt.
- LCD interface, keypad decoder, RS232 serial port, 8-bit ADC, four relay driver outputs, four buffered inputs.
- Power with 5VDC regulated or 6-13 VDC unregulated
- Large prototyping area, processor pins routed to headers
- Optional enclosures, keypads, LCDs, everything you need
- BCI51 Pro BASIC Compiler w/50+ Dallas keywords $399

SYSTRONIX® TEL: 801.534.1017 FAX: 801.534.1019
555 South 300 East, Salt Lake City, UT, USA 84111

#125

little sense for any other application. Can you imagine anyone voluntarily working backwards from the keyboard output to figure out which key was pressed?

The Enhanced keyboard microcontroller can encode keys using any one of three distinct scan-code sets. You've just seen a sample of the default, Scan Code Set 2, in all its glory. If you're writing a PC keyboard-controller BIOS, that's the one you must interpret to remain compatible with everyone else.

Scan Code Set 1 is closer to the original PC/AT 84-key encoding. Although the controller tracks the shift states and sends out ersatz shift keys as needed, the codes are entirely different than Set 2. I suspect this might have been a first pass at integrating the additional keys before resolving all the backwards compatibility issues. In any case, it's now ensnared in the PC Compatibility Barnacles.

Scan Code Set 3, on the other hand, clearly shows its non-PC heritage. Unlike the two other sets, every key sends a single, one-byte make code regardless of the shift state. Break codes are two bytes long: FO followed by the key's make code. No muss, no fuss, easy to decode, and easy to use-1 like it a lot.

Also, unlike Scan Code Sets 1 and 2, many of the keys are not typematic. For example, the Ctrl and Alt keys to the right of the space bar are make only. The controller sends a single make code regardless of how long the key stays down and does not send a break code when it goes up.

Using Scan Code Set 3 in a real-mode PC application isn't an option, but in protected mode the Compatibility Barnacles don't bind us quite so tightly. Listing 2 should give you an idea of what's ahead. It lists the real-mode BIOS return values for each key alone and then with Caps-, Ctrl-, and Alt-shift. The keys are in Scan Code Set 3 order rather than the usual Set 2 hodgepodge.

Next month, we'll build a BIOS-compatible protected-mode keyboard interface without chewing through all those incomprehensible scan codes. Not only am I sure you've never seen this trick before, but I bet you never knew your keyboard had more than one scan-code set. Right?

Meanwhile, you can experiment with the keyboard in real mode using good old Debug. Set up a DOS boot diskette with the following AUTOEXEC.BAT file:

```
mode com1:9600,n,8,1
ctty com1
```

Then, fire up the a comm program on your main PC. Boot the '386SX system into DOS and type DOS and Debug commands through the serial link!

You must disable the BIOS keyboard handler before sending controller commands and reading keyboard scan codes. The easiest way is masking IRQ 1 at the 8259 interrupt controller. Read port 21 hex to get the current IMR, OR that value with 02

Listing 2—*continued*

```
KEYCODES <02827h,02822h,00000h,02800h> ; 52 41 '"
KEYCODES <>                            ; 53 52 102-key only
KEYCODES <01A5Bh,01A7Bh,01A1Bh,01A00h> ; 54 27 [{
KEYCODES <00D3Dh,00D2Bh,00000h,08300h> ; 55 13 =+
KEYCODES <08500h,08700h,08900h,08B00h> ; 56122 F11
KEYCODES <00000h,00000h,07200h,00000h> ; 57 124 Print Screen
KEYCODES <00000h,00000h,00000h,00000h> ; 58 64 Right Ctrl
KEYCODES <00000h,00000h,00000h,00000h> ; 59 57 Right Shift
KEYCODES <01C0Dh,01C0Dh,01C0Ah,01C00h> ; 5A 43 Enter
KEYCODES <01B5Dh,01B7Dh,01B1Dh,01B00h> ; 5B 28 ]}
KEYCODES <02B5Ch,02B7Ch,02B1Ch,02B00h> ; 5C 29 \| 101-key only
KEYCODES <>                            ; 5D
KEYCODES <08600h,08800h,08A00h,08C00h> ; 5E 123 F12
KEYCODES <00000h,00000h,00000h,00000h> ; 5F 125 Scroll Lock
KEYCODES <050E0h,050E0h,091E0h,0A000h> ; 60 84 Gray Down
KEYCODES <04BE0h,04BE0h,073E0h,09B00h> ; 61 79 Gray Left
KEYCODES <00000h,00000h,00000h,00000h> ; 62126 Pause
KEYCODES <048E0h,048E0h,08DE0h,09800h> ; 63 83 Gray Up
KEYCODES <053E0h,053E0h,093E0h,0A300h> ; 64 76 Gray Del
KEYCODES <04FE0h,04FE0h,075E0h,09F00h> ; 65 81 Gray End
KEYCODES <00E08h,00E08h,00E7Fh,00E00h> ; 66 15 Backspace
KEYCODES <052E0h,052E0h,092E0h,0A200h> ; 67 75 Gray Ins
KEYCODES <>                            ; 68
KEYCODES <04F00h,04F31h,07500h,00000h> ; 69 93 1 End
KEYCODES <04DE0h,04DE0h,074E0h,09D00h> ; 6A 89 Gray Right
KEYCODES <04B00h,04B34h,07300h,00000h> ; 6B 92 4 Left
KEYCODES <04700h,04737h,07700h,00000h> ; 6C 91 7 Home
KEYCODES <051E0h,051E0h,076E0h,0A100h> ; 6D 86 Gray PgDn
KEYCODES <047E0h,047E0h,077E0h,09700h> ; 6E 80 Gray Home
KEYCODES <049E0h,049E0h,084E0h,09900h> ; 6F 85 Gray PgUp
KEYCODES <05200h,05230h,09200h,00000h> ; 70 99 0 Insert
KEYCODES <05300h,0532Eh,09300h,00000h> ; 71104 . Del
KEYCODES <05000h,05032h,09100h,00000h> ; 72 98 2 Down
KEYCODES <04C00h,04C35h,08F00h,00000h> ; 73 97 5 (pad)
KEYCODES <04D00h,04D36h,07400h,00000h> ; 74102 6 Right
KEYCODES <04800h,04838h,08D00h,00000h> ; 75 96 8 Up
KEYCODES <00000h,00000h,00000h,00000h> ; 76 90 Num Lock
KEYCODES <0E02Fh,0E02Fh,09500h,0A400h> ; 77 95 Gray /
KEYCODES <>                            ; 78
KEYCODES <0E00Dh,0E00Dh,0E00Ah,0A600h> ; 79108 Gray Enter
KEYCODES <05100h,05133h,07600h,00000h> ; 7A 103 3 PgDn
KEYCODES <>                            ; 7B
KEYCODES <04E2Bh,04E2Bh,09000h,04E00h> ; 7C 108 Gray +
KEYCODES <04900h,04939h,08400h,00000h> ; 70101 9 PgUp
KEYCODES <0372Ah,0372Ah,09600h,03700h> ; 7E 100 Gray *
KEYCODES <>                            ; 7F
KEYCODES <>                            ; 80
KEYCODES <>                            ; 81
KEYCODES <>                            ; 82
KEYCODES <>                            ; 83
KEYCODES <04A2Dh,04A2Dh,08E00h,04A00h> ; 84 105 Gray
```

hex, and write it back. On my system, the IMR is normally B8, so a simple 0 21 BA masks the keyboard interrupt. Obviously, you must be running Debug through the serial port when you disable the keyboard interrupt!

Check the references for the system board keyboard controller's I/O ports and bit definitions. I'll cover these in detail next month. A little advance exploration on your part will make things more comprehensible.

RELEASE NOTES

Demo Taskette 3 installs a keyboard hardware interrupt handler and displays standard system scan codes on the VGA display. Try all the Ctrl, Alt, and Shift combinations for the "gray keys" while you're at it. Be amazed at the number of scan codes for PrtSc, Break, and similar keys.

The task dispatcher now displays the number of task switches per second on the VGA's bottom line. At 33 MHz, the '386SX clocks about 400 switches per second, executing each taskette 100 times each second. Although these numbers take on more significance next month, I'd be interested to hear the results on your system.

If you didn't pick up Frank Van Gilluwe's ***The Undocumented PC*** (Addison Wesley, ISBN o-201-62277-7) last month, it's too late by now. Your fellow readers got there first. It has the best rendition of the keyboard interface I've ever seen, and other chapters are equally good. There are a few typos in the keyboard table, of course.

The tables in Hogan's ***Programmer's PC Sourcebook*** (Microsoft Press, ISBN 1-55615-321-X) summarize the various keyboards and scan code sets. The keyboard controller commands are not documented, and there are no references to the PS/2 system keyboard controller functions.

Thanks to Rick Freeman and the folks at Computer Options in Raleigh for helping me check out my ideas. I left at 6:01 Saturday evening with one each of every keyboard they had and returned at 9:59 Monday. It was quite a Sunday!

Next month, we dive back into heavy-duty PM coding. Without this month's background, it won't make any sense at all!

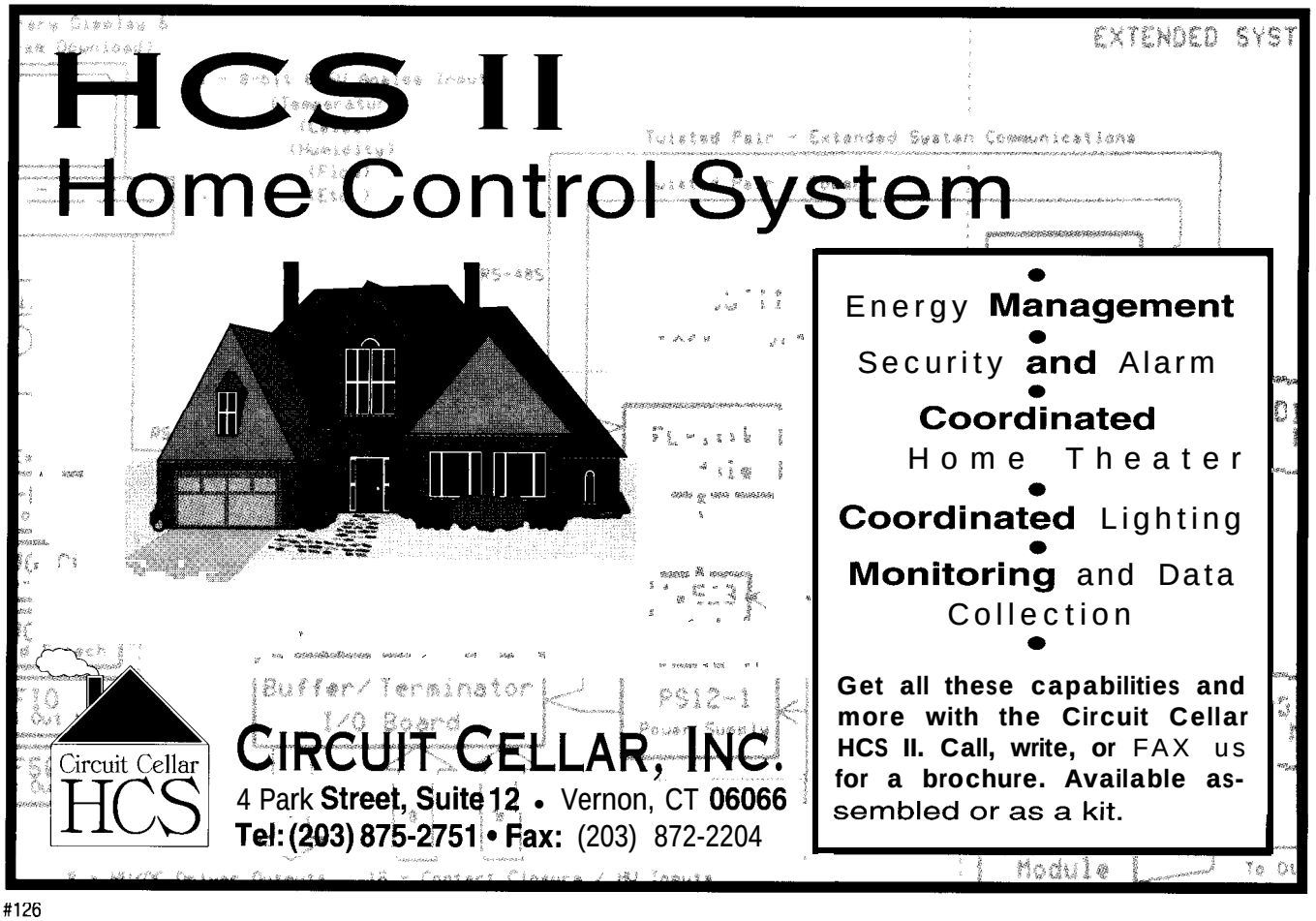
Ed Nisley, as Nisley Micro Engineering, makes small computers do amazing things. He's also a member of Circuit Cellar INK's engineering staff. You may reach him at *ed.nisley@circellar.com* ***or*** 74065.13638 *compuserve.com.*

SOFTWARE

Software for this article is available from the Circuit Cellar BBS and on Software On Disk for this issue. Please see the end of "ConnecTime" in this issue for downloading and ordering information.

I R S

413 **Very Useful**
414 Moderately Useful
415 Not Useful



#126