INT9jr

Like most other jr owners, I was dismayed when I got my update copy of Microsoft QuickBasic 3.0 and found it typing characters twice while I was using the built-in editor. I had heard that the jr "would not be supported" by QB 3.0, but they said the same thing about 2.0, and I use that all the time. So, I went to see what the trouble was.

The big symptom was the character was being entered on both the make and break of the keystroke. Ha, bet I caught you with terminology there! The "make" of a keystroke is simply the key being pressed down, the "break" is when the key is released. The hardware treats these as two different events, telling the computer a different thing for the make and break of the same key (the difference is that the make scan code's high bit is low, while the break's is high). This to me meant that the break keystroke was losing its high bit before it got processed by the INT 9H routine, which translates scan codes into characters. That was also why the shift keys got stuck in the QB3 editor; the computer never saw the break of the shift key, so it assumed you were always holding it down.

Well, fellow jr lovers, it turns out that the fault in this matter is the jr's, at least the jr ROM BIOS. Normal INT 9H on a PC and compatible gets the scan code to process from a "port". When you read this port, it does not change what is there (it's a "non-destructive read"). Thus, when INT 9H is grabbed, the grabbing procedure can read from the port and know that it won't affect anything farther down the INT 9 chain, because they should read from the port, too, and get the same value. It's a valid assumption, and MicroSoft assumed it for the QB3 editor's INT 9H grabbing strategy.

But it doesn't hold for the jr. In order to make the jr INT 9 level compatible with the PC, the jr BIOS uses a three-tiered approach. First, a software interrupt translates the infrared signals into a 62-key scan code (this happens to be the NMI interrupt, more on that later). A second software interrupt then translates this 62-key code into an IBM 83-key code, and then passes it on to the third software interrupt, INT 9H. This makes the INT 9H code very similar between the two machines, plus gives the possibility of adding an 83-key keyboard to the jr. In software, you would just bypass the first two interrupts and go to the third. (If you don't believe me, look at the ROM diagnostics with Ctrl-Alt-Ins. There are *two* keyboard diagnostics, one for 62 keys, the other for 83 keys.)

BUT, even though the second level software interrupt does a write to the port, the jr INT 9H procedure does NOT do a read from the port. It assumes the scan code will be coming in the AL register. When INT 9H directly follows the second level interrupt, that is the case. BUT, if something else has grabbed INT 9H, it may not put the scan code into the AL register before passing it on. This is probably why the early version of SideKick, an INT 9H grabber, didn't work on the jr. In QB3's case, what was in the AL register before it got to the ROM BIOS to be decoded was the scan code with the high bit masked off, which explains everything.

The fix is so simple, it's a wonder why Microsoft didn't offer it on the QB disk package and tell jr owners "Do this to get it to work." All you have to do is have a TSR program right ahead of BIOS in the INT 9H chain, have it read the port and put the value in the AL register, and then pass control to BIOS. When QB3 comes along, it grabs INT 9H from the TSR routine, not BIOS. When control eventually gets to BIOS, the TSR routine has put things in the right place.

Well, I wrote it! I call it INT9JR, and I've included the assembly source and the .COM file with this ARC. Just put it before any other INT 9H grabbing routine, so it is always followed by BIOS. This typically means putting the line

```
INT9JR
```

as close to the beginning of AUTOEXEC.BAT as possible.

For those of you out there who would rather have a device driver that does the same thing instead of a TSR, I include INT9SYS. To use it, put the line

```
DEVICE=INT9SYS.SYS
```

in your CONFIG.SYS file. You should use either INT9JR or INT9SYS, but not both (though using both won't hurt).
I've done this, and tried to test it as much as I can. It hasn't interfered with normal operations, SideKick, comm programs, or editors, and it certainly makes QB3 work as advertised. But if anybody finds any problems, let me know.

For my next trick .... That NMI thing associated with the jr keystrokes is the main reason why programs that use the 8087 math chip do not typically work on jrs equipped with 8087 daughterboards, like TIAC's (the one I have). Normal PCs have the 8087 use the NMI to signal errors in math, like divide-by- zero. The program can then grab the NMI to capture such errors without having to check for them in software, making the program faster. Normally, nothing else uses NMI, so most programs that grab NMI don't make provisions to pass it on. So, on the jr, if a program does that, it grabs the interrupt that enables you to type, basically locking the system if the program wants you to type something at it. I developed a routine to fix this problem on Microsoft's mainline languages (C, Fortran, Pascal), and called it JR87OEM.ARC in the Hardware DL on the CompuServe PC Junior Forum. For QB87, the 8087 version of QB3, that fix isn't applicable, so I'm going to have to come up with a new one. Trust me, I know what I'm doing.

John Bongiovanni
CompuServe
70137,2401
PCjr Conference
Co-Sysop
Lunacy PC Board
BBS
(818) 894-1248

```
;       PCjr INT 9, keyboard interrupt, fix device driver
;
; first, the interrupt area
;
kbint           equ     09H
oseg    segment at 0H
        org 4*kbint
i9off           dw      ?
i9seg           dw      ?
oseg    ends
CSEG    SEGMENT PARA PUBLIC 'CODE'
        org 0h
;
XDV             PROC FAR
                ASSUME CS:CSEG,DS:CSEG,ES:CSEG
BEGIN:
START           EQU $
;  Header for DOS Device Drivers
NEXT_DEV        DW  -1          ; fake pointer to next device
                                  driver
                dw -1
ATTRIBUTE       DW  08000H      ;character device with IOCTL
                                 capability
STRATEGY        DW  XDV_STRAT   ;pointer to function which queues
                                 request header
FUNC_CALL       DW  XDV_FUNC    ;pointer to operating functions
                                 switch
DEV_NAME        DB  '&JRINT9%'  ;8-byte device name field
;
;  Pointer to function request from DOS
RH_OFF          DW  ?
RH_SEG          DW  ?
;
;  Device Strategy - set pointer to request header from DOS
XDV_STRAT:      MOV  CS:RH_SEG,ES
                MOV  CS:RH_OFF,BX
                RET
;
;  Device Interrupt Handler
XDV_FUNC:       ;preserve machine state
                PUSHF
                CLD
                PUSH  DS
                PUSH  ES
                PUSH  AX
                PUSH  BX
                PUSH  CX
                PUSH  DX
                PUSH  DI
                PUSH  SI
;       Set DS to CS value
                PUSH  CS
                POP   DS
;       Load ES and BX with RH_SEG and RH_OFF
                LES   BX,DWORD PTR CS:RH_OFF
;       Branch to INIT ONLY
                MOV   AL,ES:[BX+2]      ; get function code byte
                OR    AL,AL             ;INIT?
                JNZ   DONE              ;NO, DO NOTHING
```

```
 ;  Device Initialization
 INIT:
                     MOV     AL,0            ;zero al
                     mov     dx,0a0h         ;NMI port
                     OUT     dx,AL           ;disable NMI
                     CLI                     ;disable interrupts
                     xor     AX,AX           ;zero ax
                     MOV     DS,AX           ;point to interrupt area
           assume ds:oseg
                     MOV     ax,i9off   ;save old int9                  [0024]
                     MOV     cs:old9off,ax
                     MOV     ax,i9seg
                     MOV     cs:old9seg,ax
                     MOV     ax,offset new9  ;set new int9
                     MOV     i9off,ax
                     MOV     ax,cs
                     MOV     i9seg,ax
                     sti                     ;enable interrupts
                     MOV     AL,080H         ;al=80
                     mov     dx,0a0h         ;nmi port
                     OUT     dx,AL           ;enable nmi
                     MOV     ax,OFFSET LASTWORD       ;end offset of XDV
                     MOV  ES:[BX+14],ax      ;dx:0 points to end of
                                              buffer
                     MOV  ES:[BX+16],cs
;
done:                MOV  WORD PTR ES:[BX+3],0100H ;DONE, NOERROR
;   Restore registers and exit
                     POP  SI
                     POP  DI
                     POP  DX
                     POP  CX
                     POP  BX
                     POP  AX
                     POP  ES
                     POP  DS
                     POPF
                     RET
;
XDV                  ENDP
;
kbport   equ     060H
old9     label  dword                        ;where to store ...
old9off dw ?                                 ; old 9 stuff
old9seg dw ?
NEW9     PROC     FAR                        ;new int 9
         in al,kbport
         jmp dword ptr cs:old9
NEW9     ENDP
;
LASTWORD            label byte         ;end of XDV - used for TSR
CSEG                ENDS
                    END   BEGIN
```

INT 9 FIX FOR JR - .COM VERSION

```
         kbport equ 060H                          ;keyboard port
CODE     SEGMENT
         ORG      100H
         ASSUME   CS:CODE
ENTRY:   JMP      INIT
old9     label dword                              ;where to store ...
old9off  dw ?                                     ; old 9 stuff
old9seg  dw ?
NEW9     PROC     FAR                             ;new int 9
         in al,kbport
         jmp dword ptr cs:old9
NEW9     ENDP
;
top      label byte                               ;where to TSR
;
INIT:    mov ax,3509H                             ;get interrupt 9
         int 21H
         mov cs:old9off,bx                        ;save offset
         mov cs:old9seg,es                        ;save segment
         push cs                                  ;move cs
         pop ds                                   ;to ds
         mov dx,offset new9                       ;offset in dx
         mov ax,2509H                             ;set int 9
         int 21H
;
         MOV ax,OFFSET top
         mov cl,4
         shr ax,cl
         inc ax
         mov dx,ax
         mov ax,3100H
         INT 21H
         RET
CODE     ENDS
         END      ENTRY
```