```lisp
(setf *keyb01*

'(let ()


(defconstant VK_L_CTRL 1310720)
(defconstant VK_L_ALT 1114112)
(defconstant VK_R_ALT −535756800)
(defconstant VK_R_CTRL −535560192)
(defconstant VK_L_SHFT 1179648)
(defconstant VK_R_SHFT 5832704)
(defconstant VK_CAPS 5767168)
(defconstant VK_ENTER 5898240)
(defconstant VK_TAB 851968)
(defconstant VK_BKSP 6684672)
(defconstant VK_ESC 7733248)
(defconstant VK_F1 327680)
(defconstant VK_F2 393216)
(defconstant VK_F3 262144)
(defconstant VK_F4 786432)
(defconstant VK_F5 196608)
(defconstant VK_F6 720896)
(defconstant VK_F7 8585216)
(defconstant VK_F8 655360)
(defconstant VK_F9 65536)
(defconstant VK_F10 589824)
(defconstant VK_F11 7864320)
(defconstant VK_F12 458752)
(defconstant VK_INSERT −529530880)
(defconstant VK_HOME −529793024)
(defconstant VK_PG_UP −528678912)
(defconstant VK_DELETE −529465344)
(defconstant VK_END −529989632)
(defconstant VK_PG_DN −528875520)
(defconstant VK_U_ARROW −529203200)
(defconstant VK_L_ARROW −529858560)
(defconstant VK_D_ARROW −529399808)
(defconstant VK_R_ARROW −529268736)


;;
;; ps2 scancode to ascii tables
;;
;;



(setq *unmodif-table* (vector 256 256 256 256 256 256 256 256 256 256 256 256 25
6 9 94 256
 256 256 256 256 256 113 49 256 256 256 121 115 97 119 50 256
 256 99 120 100 101 52 51 256 256 32 118 102 116 114 53 256
 256 110 98 104 103 122 54 256 256 256 109 106 117 55 56 256
 256 44 107 105 111 48 57 256 256 46 45 108 246 112 223 256
 256 256 228 256 252 180 256 256 256 256 13 43 256 35 256 256
 256 60 256 256 256 256 256 256 256 256 256 256 256 256 256 256
 256 256 256 256 256 256 27 256 256 256 256 256 256 256 256 256
 256 256 256 256 256 256 256 256 256 256 256 256 256 256 256 256))

(setq *altgr-table* (vector 256 256 256 256 256 256 256 256 256 256 256 256 256
9 176 256
```



```lisp
 256 256 256 256 256 81 33 256 256 256 89 83 65 87 34 256
 256 67 88 68 69 36 167 256 256 32 86 70 84 82 37 256
 256 78 66 72 71 90 38 256 256 256 77 74 85 123 91 256
 256 59 75 73 79 125 93 256 256 58 95 76 214 80 92 256
 256 256 196 256 220 96 256 256 256 256 13 126 256 39 256 256
 256 124 256 256 256 256 256 256 256 256 256 256 256 256 256 256
 256 256 256 256 256 256 27 256 256 256 256 256 256 256 256 256
 256 256 256 256 256 256 256 256 256 256 256 256 256 256 256 256))

(setq *shft-table* (vector 256 256 256 256 256 256 256 256 256 256 256 256 256 9
 176 256
 256 256 256 256 256 81 33 256 256 256 89 83 65 87 34 256
 256 67 88 68 69 36 167 256 256 32 86 70 84 82 37 256
 256 78 66 72 71 90 38 256 256 256 77 74 85 47 40 256
 256 59 75 73 79 61 41 256 256 58 95 76 214 80 63 256
 256 256 196 256 220 96 256 256 256 256 13 42 256 39 256 256
 256 62 256 256 256 256 256 256 256 256 256 256 256 256 256 256
 256 256 256 256 256 256 27 256 256 256 256 256 256 256 256 256
 256 256 256 256 256 256 256 256 256 256 256 256 256 256 256 256))


;;
;; auxiliary routines
;;


(defun p-shiftl-8 (x)
  (p-shiftl x 8))

(defun not (x)
  (if x nil 1))


;;
;; keyboard interface routines
;;
;;


(defun next-i (i m)
  (addr-and (%+ i 1) m))

(defun mod-m (i m)
  (if (%>= i m)
      (%- i m)
    i))


;;
;; irq puts characters into a fifo
;; get pointer - located at (100 0)
;; insert pointer - located at (100 4)
;;

(setq base_ctrl (p-constr 57344 0))

;;
;; (get-word base_ctrl 8) after irq produces 00 00 00 xx where xx is byte value
sent
;; from keyboard controller
;;
;;
```

```lisp
;; state =
;; 0 wait char event
;; 1 received e0 after wait
;; 2 received f0 after wait
;; 3 received f0 after e0

;; e0 = 224
;; f0 = 240


(defconstant *e0* 224)
(defconstant *f0* 240)


(defconstant *shft* 1)
(defconstant *ctrl* 2)
(defconstant *alt* 4)
(defconstant *alt-gr* 8)

(defconstant *setcode* 16)


(defconstant *shft-up* -2)
(defconstant *ctrl-up* -3)
(defconstant *alt-up* -5)
(defconstant *alt-gr-up* -9)

(defconstant *resetcode* -17)


(let ((state 0)
      (modif-state 0)
      (new-char-val 0)
      (new-event-ready-flag nil))

  (defun ch-to-int (ch)
    (addr-and (p-shiftl-1 (p-shiftl-1 ch)) 255))

  (defun process-key-event (ch)
    (setq ch (ch-to-int ch))
    (cond ((%= state 0)
           (setq new-event-ready-flag nil)
           (cond ((%= ch *e0*)
                   (setq state 1))
                  ((%= ch *f0*)
                   (setq state 2))
                  (t (progn
                       (akt-char-sr 1 0 ch)
                       (akt-modifier-sr 1 0 ch)
                       (setq state 0)))))
          ((%= state 1)
           (cond ((%= ch *f0*)
                   (setq state 3))
                  (t (progn
                       (akt-char-sr 1 *e0* ch)
                       (akt-modifier-sr 1 *e0* ch)
                       (setq state 0)))))
          ((%= state 2)
           (progn
             (akt-char-sr 0 0 ch)
```



```lisp
             (akt-modifier-sr 0 0 ch)
             (setq state 0)))
          ((%= state 3)
           (progn
             (akt-char-sr 0 *e0* ch)
             (akt-modifier-sr 0 *e0* ch)
             (setq state 0)))))


;;
;; sr = 1 : set, sr = 0 : release
;;

  (defun akt-char-sr (sr x c)
    (setq new-event-ready-flag 1)
    (setq new-char-val
          (%+ (p-constr (%+ (p-shiftl-8 x) c) 0)
              (if (%= x 0)
                  (cond ((%/= (addr-and modif-state *shft*) 0)
                          (%svref *shft-table* c))
                         ((%/= (addr-and modif-state *alt-gr*) 0)
                          (%svref *altgr-table* c))
                         (t (%svref *unmodif-table* c)))
                256))))

  (defun akt-modifier-sr (sr x c)
    (if (%= sr 1)
        (setq modif-state
              (addr-ior *setcode*
                        (cond ((%= x *e0*)
                               (cond ((%= c 17) ;; AltGr
                                       (addr-ior modif-state *alt-gr*))
                                      ((%= c 20) ;; Ctrl_R
                                       (addr-ior modif-state *ctrl*))
                                      (1 modif-state)))
                              (1 (cond ((or (%= c 18) (%= c 89)) ;; Shift_LR
                                         (addr-ior modif-state *shft*))
                                        ((%= c 17) ;; Alt_L
                                         (addr-ior modif-state *alt*))
                                        ((%= c 20) ;; Ctrl_L
                                         (addr-ior modif-state *ctrl*))
                                        (1 modif-state))))))
      (setq modif-state
            (addr-and *resetcode*
                      (cond ((%= x *e0*)
                             (cond ((%= c 17) ;; AltGr
                                     (addr-and modif-state *alt-gr-up*))
                                    ((%= c 20) ;; Ctrl_R
                                     (addr-and modif-state *ctrl-up*))
                                    (1 modif-state)))
                            (1 (cond ((or (%= c 18) (%= c 89)) ;; Shift_LR
                                       (addr-and modif-state *shft-up*))
                                      ((%= c 17) ;; Alt_L
                                       (addr-and modif-state *alt-up*))
                                      ((%= c 20) ;; Ctrl_L
                                       (addr-and modif-state *ctrl-up*))
                                      (1 modif-state)))))))



  (defun new-event-ready ()
    new-event-ready-flag)
```

```lisp
  (defun new-char ()
    new-char-val)

  (defun new-modifier ()
    modif-state)

)

(defconstant *char-queue* 112)
(defconstant *modif-queue* 148)

(defun irq (x)
  (let ((ch (get-word base_ctrl 8))
        (get-pt (get-word 100 0))
        (insrt-pt (get-word 100 4)))

    (process-key-event ch)
    (if (new-event-ready)
        (progn
          (set-word *char-queue* insrt-pt (new-char))
          (set-word *modif-queue* insrt-pt (new-modifier))

          (let ((insrt-pt1 (next-i insrt-pt 31)))

            (if (not (%= insrt-pt1 get-pt))
                (set-word 100 4 insrt-pt1)))))

    (let ((s (get-status)))
      (set-status (addr-ior s 128))
      x)))


;;
;; at (0 16) is the pointer to the interrupt routine
;;

(set-word 100 0 0)
(set-word 100 4 0)

(set-word (p-constr 0 16) 0 (%symbol-function 'irq))

;;
;; the print routines
;;


(defun to-ascii (val)
  (if (%<= val 9)
      (%+ val 48)
    (%+ val 55)))

(defun print-hex-digit (pos val)
  (set-byte base_vga (p-shiftl pos 2) (to-ascii val)))

(defun print-hex-byte (pos val)
  (print-hex-digit (%+ pos 1) (addr-and val 15))
  (print-hex-digit pos (addr-and (tag-and (p-shiftr val 4) 0) 15)))


(defun print-hex (pos val)
  (let ((v val))
```



```lisp
    (print-hex-byte (%+ pos 6) (addr-and (p-shiftl val 2) 255))
    (print-hex-byte (%+ pos 4) (addr-and (setq v (tag-and (p-shiftr v 6) 0)) 255
))
    (print-hex-byte (%+ pos 2) (addr-and (setq v (tag-and (p-shiftr v 8) 0)) 255
))
    (print-hex-byte pos (addr-and (tag-and (p-shiftr v 8) 0) 255))))


(setq base_vga (p-constr 61440 0))


;;
;; the original (getch) routine
;;


;;;;  (defun getch ()
;;;;    (let ((getpt (get-word 100 0))
;;;;          (insrtpt (get-word 100 4)))
;;;;      (if (%= getpt insrtpt)
;;;;          -1
;;;;        (let ((ch (get-word 110 getpt)))
;;;;          (set-word 100 0 (next-i getpt 15))
;;;;          ch))))

(defun update-getpt (getpt)
  (set-word 100 0 (next-i getpt 31)))

(defun main ()
  (main-init)
  (setq *i* 0)
  (do-until (nil)
            (print-hex 5360 (setq *i* (%+ *i* 1)))
            (let ((getpt (get-word 100 0))
                  (insertpt (get-word 100 4)))
              (if (%/= getpt insertpt)
                  (main-dispatch getpt)))))

(defun main-init ()
  1)

;;
;; memdiffs for movement on screen
;;

(defconstant *xdiff* 4)
(defconstant *ydiff* 532)

(defconstant *line-len* 133)

(let ((xpos 0)
      (ypos 0)
      (xpos-mem 0)
      (ypos-mem 0))

(defun main-dispatch (getpt)
  (let ((ch (get-word *char-queue* getpt))
        (modif (get-word *modif-queue* getpt)))

    (if (and (%= (addr-and ch 256) 0) (%/= (addr-and modif *setcode*) 0))
        (let ()
          (set-byte base_vga (%+ ypos-mem xpos-mem) (addr-and ch 255))
```

```
          (if (%< xpos (%- *line-len* 1))
              (let ()
                (setq xpos (%+ xpos 1))
                (setq xpos-mem (%+ xpos-mem *xdiff*)))
            (let ()
              (setq xpos 0)
              (setq xpos-mem 0)
              (setq ypos (%+ ypos 1))
              (setq ypos (%+ ypos *ydiff*))))))

    (print-hex 5320 modif)
    (update-getpt getpt)))

)

(%print 1212)
(%print (p-shiftl 111 2))
(main)))
```