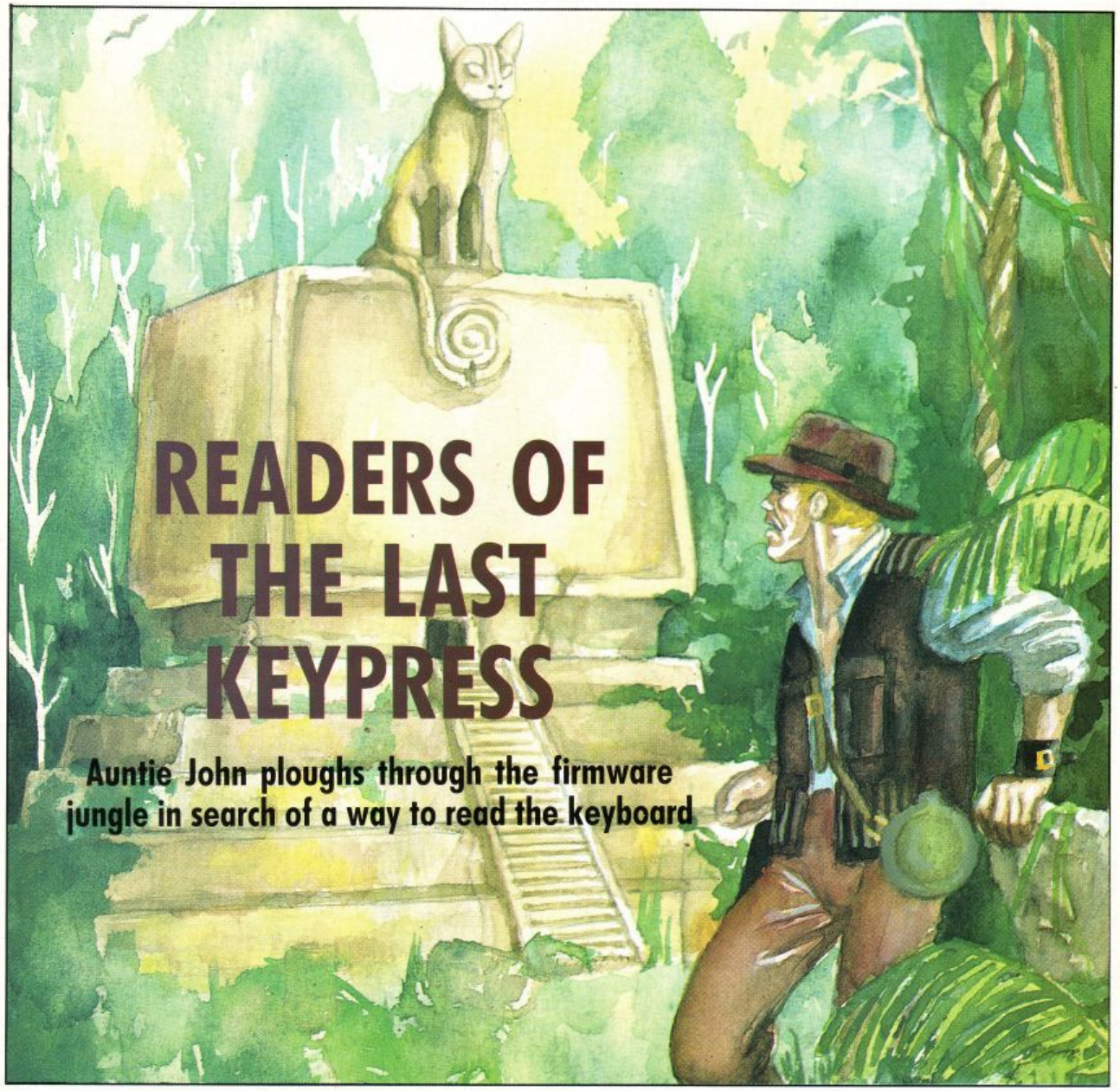

READERS OF THE LAST KEYPRESS

Auntie John ploughs through the firmware jungle in search of a way to read the keyboard

AFTER last month's look at writing things to the screen, it seems only natural to spend some time looking at how to read things from the keyboard. I know if you use your computer a lot you might find that difficult because the letters start to wear off the keytops, but that's easily solved with a hot pin and a book called How to read Braille.

The type of reading I was actually referring to was scanning the keyboard using software to see which keys are being pressed.

There are two main ways to check the keyboard for keypresses, namely real-time and not real-time. If you are writing an arcade game then obviously you will need a realtime check on the keys, and you will also need to be able to read more than one key at a time so you can go left and nuke-the-furry-little-gibbles at (almost) the same time.

If you are writing a text input routine, then you can use the not real-time version, because the computer doesn't need to be doing anything when it is waiting for you you to press a key.

We are lucky because firmware routines to do both are just sitting in rom waiting for us to call them.

Incidentally, did you know the keyboard is scanned using the input/output port of Arnold's sound chip? Course you did, silly me. Once I connected about 20 metres of doorbell extension cable to my joystick port to make something that I could use in a quiz game. Each competitor had a switch wired to the computer and was checked for by a program that decided who pressed their switch first. All was going fine until the very first quiz when, five minutes before the start, the computer went Neeekkkk! Phut.... and died.

With all that capacitance about, the cable I'd connected decided to build up a charge and zap my little sound chip. Sniff.

So out with the soldering iron and sweetie cigarettes, and me and my mate Green unsoldered the soundchip and put another one in. There is a moral to this story but I can't remember it. Something to do with buffering inputs to the joystick port probably.

"What about scanning the keyboard?" I pretend to hear you cry. And why not?

A casual, almost cursory, and decidedly cool glance at the Amsoft Firmware Guide (available from all good stockists) will provide you with at least five calls that look feasible. My favourite ones are 2:KM WAIT CHAR (#BB06) which returns an Ascii code in the A register, and 10:KM TEST KEY (#BB1E) which when given the key number (not the Ascii) sets the zero flag false if the key is pressed and true if it isn't.

I won't insult your intelligence by telling you which is the realtime, and which waits. As there are two different calls, two examples should suffice to enable you to use them in your own programs. Both use subroutines you may (or may not) find useful in later life.

Figure I: Psuedo-code for Program I

```
LET HL=WELCOME STRING       ;HL points to welcome message using
CALL PRINT_STRING           ;last month's string print routine.
SET COUNTER=0               ;Set loop counter to zero.
LOOP
CALL WAIT_KEY               ;Firmware routine that returns key in A.
IS A=CARRIAGE RETURN?       ;Has Return been pressed?
  YES, goto EXIT            ;Yes, finish
  NO, carry on
CALL TXT_OUTPUT             ;Print the character
LET (HL)=A                  ;Poke the character into memory.
LET HL=HL+1                 ;Increment HL to point at next storage location.
LET COUNTER=COUNTER+1       ;Increment counter.
IS COUNTER=80?              ;Check for 80 characters.
  NO, goto LOOP             ;No, go back.
  YES, carry on
EXIT                        ;The end of the routine.
LET (HL)=0                  ;Poke in an end of text marker.
{E}
```

Figure II: Psuedo-code for Program II

```
{L}
MODE 1
LET XPOS=20                 ;Clear the screen to 40 columns.
LET FLAG=0                  ;Point XPOS to middle of the screen.
                            ;Reset variable that is used to check
                            ;for the Escape key being pressed.
;Routines for moving left
IS XPOS=1?
  YES, skip to move right routine
  NO, is Z pressed? If YES then erase blob:XPOS=XPOS-1:reprint blob
;Routines for moving right
IS XPOS=40?
  YES, skip to ESC
  NO, is X pressed? If YES then erase blob:XPOS=XPOS+1:reprint blob
;Routine to check Escape key
Is Escape pressed?
  YES, LET FLAG=1
IS FLAG=0?
  YES, goto LOOP
  NO, return to Basic
{E}
```

Program I

```
{L}
;
;Wait Key example.
;
;Entry and display of text strings using firmware routines.
;

org &4000                   ;Code starts at this address.

wait_key equ &bb18          ;Firmware, waits for a keypress.
txt_output equ &bb5a        ;Firmware, displays Ascii char held in A.


;Display the message.

ld hl,welcome               ;String which is defined later on.
call print_string           ;User defined subroutine.


;Read in text from keyboard.

ld hl,buffer                ;Some specially reserved memory.
call input_string           ;Another subroutine.


;Display the entire string, but in inverse.

ld hl,setinks               ;Points to some characters that will swap
call print_string           ;pen and paper, and so display in inverse.
ld hl,buffer                ;Points to the text typed in.
call print_string           ;Display it.
ld hl,setinks               ;Must repeat swapping paper and pen before
call print_string           ;before finishing. What if we didn't?
ret                         ;Return to Basic.



;
;*** Subroutines ***
;


;This routine takes a string pointed to by HL and displays it on the screen.
;The string can be any length, but must end with the null character 0.
;The firmware routine txt_output is used. Registers HL and A are corrupted.

ld a,(hl)                   ;Get character pointed to by HL.
cp 0                        ;Is it 0? (the end marker)
ret z                       ;If it is, then return from subroutine.
call txt_output             ;If not, carry on and display the character.
inc hl                      ;Move HL to point to next character.
jr psloop                   ;Jump back.


;This routine reads a string character by character from the keyboard,
;echoes it to the screen and puts it into a block of memory pointed to
;by HL. The entry is terminated if either Return is pressed, or the
;number of characters exceeds 80. An end of string marker (zero) is
;inserted at the end of the text.
;
;The firmware routines txt_output and wait_key are used.
;Registers HL, B and A are corrupted.

ld b,0                      ;Set loop counter.
call wait_key               ;Get key pressed.
cp 13                       ;Is it Return?
jr z,exit                   ;Yep, end routine.
call txt_output             ;Else display character.
ld (hl),a                   ;Poke it into buffer.
inc hl                      ;Point HL to next location.
inc b       ;End of the program.
{E}
```

Program II

```
{L}
;
;Test Key example.
;
;Moving a blob using the firmware routines.
;

org &4000                   ;start of code

test_key equ &bb1e          ;Firmware routine to check a key.
txt_output equ &bb5a        ;The old favourite text output routine.
set_mode equ &bc0e          ;just like a Basic MODE command.
set_cursor equ &bb75        ;just like a Basic LOCATE.

ld a,1                      ;We want mode 1 (take my word for it).
```

Km Wait Char

This is the key routine that waits for you to press a key (drat, I've given it away!). The example program uses the routine to allow the user to type in a string of Ascii characters from the keyboard, store them in memory, and then print them out again.

Exciting, you think. Yes I know, real cutting-edge stuff, but the routines could be used in an adventure game, or to input a name for the high score in your latest epic.

Program I is listed in standard Z80 mnemonics (pronounced ne-nom-mon-nom-om-om-om-ix) which you must type into an Assembler to make any use of. I know we've had this conversation before, but to learn machine code *you need an assembler*.

Figure I shows the psuedo-code for Program I, so you can see what's going on. There are some points to look out for. The normal cursor is not displayed on the screen, which makes it look a bit odd, and the Delete and Escape keys don't do anything except make little patterns appear. You would have to write another routine to check for these in the same way that code 13 (the Return key) is checked for. The computer keeps waiting for characters until either 80 have been typed in, or Return is pressed.

Km Test Key

As you would most likely use this routine when writing an exciting, if-it-moves-kill-it game, the example in Program II shows you how to use this, and other, firmware routines to move a blob on the screen from left to right. I expect you will see it in the arcades soon.

Figure II shows the pseudo-code for Program II. Although other firmware routines are used in this example, none of them should cause any problems to clever people like yourselves. You'll notice that the program runs much faster than anything you could write in Basic. You might like to try and slow it down by putting FRAME FLYBACK calls (call &bd19) into the main loop. This will also reduce any flicker.

Other things to try include adapting the program to run in mode 2, and trying to display more than one character at a time, say a 2 x 2 washing-machine made from User Defined Graphics.

You should also note that by changing the numbers in the check left and check right routines you can check for the joystick being used.

p;:.....hj /mnnnnnnnnnnnnnnnnnnnn, lkm ;:/.....k ccccccccccccccccccnx bvn. Sorry about that, my cat walked over the keyboard.

So, we have looked at reading and writing. What's next? Rithmetic of course! But that's next month.

```
call set_mode                ;And this call will do it.

ld a,20                      ;Set the XPOS variable to the middle
ld (xpos),a                  ;of the screen.

ld a,0                       ;This sets the FLAG variable to zero.
ld (flag),a                  ;FLAG is used to see if esc is pressed.

call print_blob              ;These are self-explanatory, aren't
call check_left
call check_right
call check_esc
ld a,(flag)                  ;Check FLAG. If it's non-zero then
cp 0                         ;esc has been pressed.
jr z,loop                    ;If it's zero, loop back.

ret                          ;Return to Basic.

;Bytes set aside to hold variables.

;
;*** Subroutines ***
;

;Check to see if blob is as far left as possible.
;Check for key number 71 (the Z key) and change XPOS if so.
;Erase and redisplay the blob.

ld a,(xpos)
cp 1
ret z                        ;if XPOS=1 then return from subroutine.

ld a,71
call test_key
ret z                        ;if key not pressed then return from s/r.

call erase_blob
ld a,(xpos)
dec a                        ;XPOS=XPOS-1
ld (xpos),a
call print_blob

ret                          ;Return from subroutine.

;Check to see if blob is as far right as possible.
;Check for key number 63 (the X key) and change XPOS if so.
;Erase and redisplay the blob.

ld a,(xpos)
cp 40
ret z                        ;if XPOS=40 then return from s/r.

ld a,63
call test_key
ret z                        ;if key not pressed then return from s/r.

call erase_blob
ld a,(xpos)
inc a                        ;XPOS=XPOS+1
ld (xpos),a
call print_blob

ret                          ;Return from s/r.

;Check for Escape key, setting (flag)=1 if pressed.

ld a,66
call test_key
ret z                        ;Return from s/r if esc not pressed.

ld a,1
ld (flag),a[0     ;Return from s/r.

;Draw a space over blob at (XPOS).

ld a,(xpos)
ld h,a
ld l,10
call set_cursor

ld a,32                      ;ascii code for a space.
call txt_output

ret                          ;Return from s/r.

end                          ;End of program.
```