Scanning the Keyboard the correct and non-KERNAL way

This routine is for the English Keyboard.

The routine can easilly be extended to handle Control Port #1 input.

The Theory

Limitations

The technical limitation on the C64 Keyboard hardware is that not more than 2 keys may be pressed at the same time if you want to be 100% sure the result is valid. In some cases, three keys will work fine but whenever 3 keys form a right angle in the keyboard scan matrix, a 4th letter will appear. The combination “ABC” will work fine but the combination “ASD” will form such a triangle and the matrix will also report that the “F” key is pressed. Same goes for “ASF” which would incorrectly read a “D” the same way.

In short, the C64 keyboard is not a piano where you can play choords and stuff.

Key Rollover

First, let's see what the definition of the term Key RollOver (KRO) is:

Falex Free Dictionary:

“**A computer keyboard circuit that allows any number of keys to be pressed in succession without having to lift a finger from any of the previous keys.**”.

Wikipedia:

“**Rollover is the ability of a computer keyboard to correctly handle several simultaneous keystrokes.**”

Oxfords:

“**a facility on an electronic keyboard enabling one or several keystrokes to be registered correctly while another key is depressed.**”

Answers.com:

“**A computer keyboard circuit that allows any number of keys to be pressed in succession without having to lift a finger from any of the previous keys.**”

PC Magazine

“**A computer keyboard circuit that allows any number of keys to be pressed in succession without having to lift a finger from any of the previous keys. Only a small number of high-end keyboards have n-key rollover. Most have 3-key rollover, which is essential for touch typing.**”

A N-Key RollOver (NKRO) can handle all keys being pressed and registered correctly. The Commodore 64 can only handle 2 keys and is therefore only capable of 2KRO.

For the purpose of this routine is to register keys being entered as correctly as possible having the “pressed in succession without having to lift a finger from any of the previous keys” in mind.

This Keyboard Routine

With the limitations and definitions in mind, this routine will accept up to 3 keys being pressed in succession as long as no shadowing (or ghosting) is produced (ie. a 4th key gets incorrectly registered).

So if you type and hold “A” + “B” + “C” (Which is a valid 3 key result and doesen't produce shadowing), it will return the keys correctly.

If you type and hold “A” + “S” + “D” (Which is an invalid 3 key result as it also will inncorrectly register the “F” key being pressed it will wait untill the “A” or “B” is released (as this will remove the shadowing) then return the correct key (“D”).

What this routine (or any other for that matter) can't do is to correctly report back 2 or 3 keys being pressed so fast that they are registered by the routine to be pressed at the exact same time. In this case, you will need to achieve a “No Activity” state before new valid input is accepted. You could in some cases have gotten away with it but it would rely on randomness which is not in thread of making a routine as secure as possible.

If you need more than 1 Alphanumeric key returned each call, this is not the routine for you.

Speed vs. Memory

The routine presented here favours speed over memory. If memory is tight, you'd wan't to loop some of the checks being carried out. However the code size including tables is roughly 2 pages in size and R-Time is pretty good.

Why not use KERNAL or the other routines on the Codebase?

KERNAL

The KERNAL keyscanner is full of bugs. You can try out the following:

- Press and hold “D”. Now press “A” → The “A” isn't recognized.
- Press and hold “D”. Now press “K” and release it again repeatedly → “D” is recognized as a new character each time “K” is released.
- The infamous Control Port #1 Bug
- Press and hold “A”+“S”+“D”. → You will get “ASF” as a result (due to key shadowing).

3-Key Rollover

Bruce Craigs 3-Key Rollover routine from C=Hacking #7 takes care of some of these matters but not all. Some bugs remains and some are introduced:

- Press and hold “A”+“B”+“C”+“D” then release the “A” → The “D” is not printed even though only 3 keys are held down (Works in KERNAL).
- Press and hold “A”+“S”+“D” → You will get “ASF” as a result (same as KERNAL).

It's simple to fix though.

Others

The other routines by Groepaz and Oswald are for all intents and purposes likely to be of more use to the average programmer than this routine as they are more compact and fast. However they lack some functionality compared to this one and is only good for single key presses (Except Oswalds routine which can handle SHIFT).

Using the routine (API)

just call the routine with a JSR and read the Carry to determine if any valid input is returned.

If the result is valid, the Accumulator will contain the Alphanumeric keys and the X & Y Registers will contain flags for all the non-Alphanumeric keys.

Return

The alphanumeric keys are returned in the Accumulator bit 0-5 (Gives us a total of 64 keys). All the non-alphanumeric keys get's returned in the X & Y Register.

Result returned in Accumulator (BIT #0 to #5)							
Code	**Descrition**	**Code**	**Descrition**	**Code**	**Descrition**	**Code**	**Descrition**
$00	@	$10	p	$20	SPACE	$30	0
$01	a	$11	q	$21		$31	1
$02	b	$12	r	$22		$32	2
$03	c	$13	s	$23		$33	3
$04	d	$14	t	$24		$34	4
$05	e	$15	u	$25		$35	5
$06	f	$16	v	$26		$36	6

Result returned in Accumulator (BIT #0 to #5)							
Code	**Descrition**	**Code**	**Descrition**	**Code**	**Descrition**	**Code**	**Descrition**
$07	g	$17	w	$27		$37	7
$08	h	$18	x	$28		$38	8
$09	i	$19	y	$29		$39	9
$0a	j	$1a	z	$2a	*	$3a	:
$0b	k	$1b		$2b	+	$3b	;
$0c	l	$1c	£	$2c	,	$3c	
$0d	m	$1d		$2d	-	$3d	=
$0e	n	$1e	Arrow Up	$2e	.	$3e	
$0f	o	$1f	←	$2f	/	$3f	

Return in X-Register							
Bit 7	**Bit 6**	**Bit 5**	**Bit 4**	**Bit 3**	**Bit 2**	**Bit 1**	**Bit 0**
CRSR UD	F5	F3	F1	F7	CRSR RL	RETURN	INST/DEL

Return in Y-Register							
Bit 7	**Bit 6**	**Bit 5**	**Bit 4**	**Bit 3**	**Bit 2**	**Bit 1**	**Bit 0**
RUN STOP	L-SHIFT	C=	R-SHIFT	CLR/HOME	CTRL		

Furthermore, the Carry is set or clear according to the following conditions:

Condition	**Carry**	**Accumulator**
No keyboard activity is detected.	Set	#$01
Control Port #1 Activity is detected.	Set	#$02
Key Shadowing / Ghosting is detected.	Set	#$03
2+ AlphaNum keys is detected within one scan	Set	#$04
Awaiting “No Activity” state	Set	#$05
No new Alphanumeric Keys detected (some key(s) being held down AND/OR some Non-Alphanumeric key is causing valid return).	Clear	#$ff
New Alphanumeric Key returned. Non-Alphanumeric keys may also be returned in X or Y Register	Clear	<>#$ff

The Routine

The code is pretty well commented but I'll go through some of the theory and thoughts behind the routine in this section.

ZERO Page

Some ZP is used to speed things up a bit.

Simple philosophy: Only variables which changes within a call is used in ZP. Things can happen outside the routine which can mess things up if we expect data in the ZP to stay alive untill next time the routine is called. So Temporary stuff in ZP only!

I will use ZP memory from $50 to $5f.

Data Direction Registers (DDRs)

Since we can forget about KERNAL in this context we have to assume we have no idea of the status of the DDRs of the ports.

(SuperCPU would be nice when setting registers like $dc02 / $dc03 😃)

Scanning for Activity

We only need to scan the keyboard matrix under certain conditions. First of all, is there any activity at all? secondly, is there any interferance from the Control Port #1?

Keyboard Activity

This is simply done by connecting all Keyboard Rows to the Port and check if the result of $dc01 is #$ff (No activity) or <> #$ff (Activity).

Control Port Activity

Control Port #1 is connected to the same Port as the Keyboard and will usually mess things up. To detect if there is any activity from the Control Port #1, we disconnect all keyboard row lines to the port and check if there is any activity.

The activity check functions to determine if we should skipp scanning the keyboard if the control port is active. The Keyboard scan routine would quickly have discover it's a non valid input by itself.

Theoretically you could could have a Control Port activation after the scan of the 5th keyboard row which would give you a faulty input (so unlikely that it hurts to think about it). We therefore do a 2nd check after the scan of the keyboard matrix to be 100% sure.

Scanning the Keyboard Matrix

In my point of view, reading the keyboard matrix is like taking a picture of a situation. How clear your picture is depends on how fast the lens can close to capture the picture. Therefore, I wan't to scan the keyboard matrix as fast as possible to get a accurate presentation of the keyboard scan matrix.

```
    //~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
    // Scan Keyboard Matrix

    lda #%11111110
    sta $dc00
    ldy $dc01
    sty ScanResult+7
    sec
    rol
```

```
    sta $dc00
    ldy $dc01
    sty ScanResult+6
    rol
    sta $dc00
    ldy $dc01
    sty ScanResult+5
    rol
    sta $dc00
    ldy $dc01
    sty ScanResult+4
    rol
    sta $dc00
    ldy $dc01
    sty ScanResult+3
    rol
    sta $dc00
    ldy $dc01
    sty ScanResult+2
    rol
    sta $dc00
    ldy $dc01
    sty ScanResult+1
    rol
    sta $dc00
    ldy $dc01
    sty ScanResult
```

You can ofcourse choose to use a loop if memory is your concern.

As you can see, I have chosen not to “DeBounce” the reading of the port. Debouncing keys is handled by the routine by clever buffer usage and if a key is detected or released this or next scan matters little as the routine will still read the keyboard correct. I may be wrong about this but untill proven otherwise, this is the aproach I'll stick to.

Scanning for non-Alphanumeric Keys

This is easilly done by doing some arithmetics on the Scan Result.

For the result returned in X, you simply need to read the 8th line of the ScanResult as it's already ligned up perfectly for us.

For the return in the Y-Register, some arithmic trickery is needed (check source code).

A couple of ZP Temporary storage is used which then simply is ready into the X & Y-Register upon exit of the routine.

Deciphering the Keyboard Scan Matrix

I unrolled this part of the Routine as the looped version suxored R-Time. It works as follows:

1. Is there any keys pressed in the current row? → NO: Check Next Row
2. YES: Use X-Register as a pointer index into a key-table and arithmically shift the values in the scan untill you find the keys which are activated
3. Once a key as found, check for overflow (max 3 keys pr. scan is allowed due to keyboard bugs)
4. Store key in BufferNew

Would definately be handy with an extra register here

Buffers

This is where the magic happens. The BufferNew is compared agains BufferOld to check if there are any new keys present (This takes care of the DeBounce issue). If there is, they get added to Buffer which is the output buffer (holding up to 3 keys as well^^) which then in turn, cronologically returns the keys to the caller. It's kinda hard to wrap ones head about dealing with several buffers and such. For me, trial and error combined with a good test program helped a lot when coding this.

Known Issues

At the moment, none that I am aware of.

The Code

In good old KickAss Format.

```
/*~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
~~~~
    Keyboard IO Routine
    ~~~~~~~~~~~~~~~~~~~
        By: TWW/CTR


    Preparatory Settings
    ~~~~~~~~~~~~~~~~~~~~
        None


    Destroys
    ~~~~~~~~
        Accumulator
        X-Register
        Y-Register
```

```
        Carry / Zero / Negative
        $dc00
        $dc01
        $50-$5f


    Footprint
    ~~~~~~~~~
        #$206 Bytes


    Information
    ~~~~~~~~~~~
        The routine uses "2 Key rollower" or up to 3 if the key-combination
doesen't induce shadowing.
        If 2 or 3 keys are pressed simultaneously (within 1 scan) a "No
Activity" state has to occur before new valid keys are returned.
        RESTORE is not detectable and must be handled by NMI IRQ.
        SHIFT LOCK is not detected due to unreliability.


    Usage
    ~~~~~
      Example Code:

            jsr Keyboard
            bcs NoValidInput
                stx TempX
                sty TempY
                cmp #$ff
                beq NoNewAphanumericKey
                    // Check A for Alphanumeric keys
                    sta $0400
            NoNewAphanumericKey:
            // Check X & Y for Non-Alphanumeric Keys
            ldx TempX
            ldy TempY
            stx $0401
            sty $0402
       NoValidInput:  // This may be substituted for an errorhandler if
needed.


    Returned
    ~~~~~~~~

        +=================================================+
        |             Returned in Accumulator             |
        +===========+===========+============+===========+
        |  $00 - @  |  $10 - p  |  $20 - SPC  |  $30 - 0  |
```

```
        |  $01 - a  |  $11 - q  |  $21 -       |  $31 - 1  |
        |  $02 - b  |  $12 - r  |  $22 -       |  $32 - 2  |
        |  $03 - c  |  $13 - s  |  $23 -       |  $33 - 3  |
        |  $04 - d  |  $14 - t  |  $24 -       |  $34 - 4  |
        |  $05 - e  |  $15 - u  |  $25 -       |  $35 - 5  |
        |  $06 - f  |  $16 - v  |  $26 -       |  $36 - 6  |
        |  $07 - g  |  $17 - w  |  $27 -       |  $37 - 7  |
        |  $08 - h  |  $18 - x  |  $28 -       |  $38 - 8  |
        |  $09 - i  |  $19 - y  |  $29 -       |  $39 - 9  |
        |  $0a - j  |  $1a - z  |  $2a - *     |  $3a - :  |
        |  $0b - k  |  $1b -    |  $2b - +     |  $3b - ;  |
        |  $0c - l  |  $1c - £  |  $2c - ,     |  $3c -    |
        |  $0d - m  |  $1d -    |  $2d - -     |  $3d - =  |
        |  $0e - n  |  $1e - ^  |  $2e - .     |  $3e -    |
        |  $0f - o  |  $1f - <- |  $2f - /     |  $3f -    |
        +-----------+-----------+--------------+-----------+

+============================================================================
=====
        |                             Return in X-Register
|
+=========+=========+=========+=========+=========+=========+=========+=====
====+
        |  Bit 7  |  Bit 6  |  Bit 5  |  Bit 4  |  Bit 3  |  Bit 2  |  Bit 1
|  Bit 0  |
        +---------+---------+---------+---------+---------+---------+-------
--+---------+
        | CRSR UD |   F5    |   F3    |   F1    |   F7    | CRSR RL | RETURN
|INST/DEL |
        +---------+---------+---------+---------+---------+---------+-------
--+---------+

+============================================================================
=====
        |                             Return in Y-Register
|
+=========+=========+=========+=========+=========+=========+=========+=====
====+
        |  Bit 7  |  Bit 6  |  Bit 5  |  Bit 4  |  Bit 3  |  Bit 2  |  Bit 1
|  Bit 0  |
        +---------+---------+---------+---------+---------+---------+-------
--+---------+
        |RUN STOP | L-SHIFT |   C=    | R-SHIFT |CLR/HOME |  CTRL   |
|         |
        +---------+---------+---------+---------+---------+---------+-------
--+---------+

        CARRY:
          - Set = Error Condition (Check A for code):
              A = #$01 => No keyboard activity is detected.
              A = #$02 => Control Port #1 Activity is detected.
              A = #$03 => Key Shadowing / Ghosting is detected.
              A = #$04 => 2 or 3 new keys is detected within one scan
              A = #$05 => Awaiting "No Activity" state
          - Clear = Valid input
              A =  #$ff => No new Alphanumeric Keys detected (some key(s)
being held down AND/OR some Non-Alphanumeric key is causing valid return).
              A <> #$ff => New Alphanumeric Key returned. Non-Alphanumeric
keys may also be returned in X or Y Register

    Issues/ToDo:
    ~~~~~~~~~~~~
        - None


    Improvements:
    ~~~~~~~~~~~~~
        - Replace the subroutine with a pseudocommand and account for
speedcode parameter (Memory vs. Cycles).
        - Shorten the routine / Optimize if possible.


    History:
    ~~~~~~~~
    V2.5 - New test tool.
           Added return of error codes.
           Fixed a bug causing Buffer Overflow.
           Fixed a bug in Non Alphanumerical Flags from 2.0.
    V2.1 - Shortened the source by adding .for loops & Updated the header
and some comments.
           Added "simultaneous keypress" check.
    V2.0 - Added return of non-Alphanumeric keys into X & Y-Registers.
           Small optimizations here and there.
    V1.1 - Unrolled code to make it faster and optimized other parts of it.
           Removed SHIFT LOCK scanning.
    V1.0 - First Working Version along with test tool.

~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
~~*/

    .pc = * "Keyboard Scan Routine"


    // ZERO PAGE Varibles
    .const ScanResult       = $50  // 8 bytes
    .const BufferNew        = $58  // 3 bytes
    .const KeyQuantity      = $5b  // 1 byte
    .const NonAlphaFlagX    = $5c  // 1 byte
    .const NonAlphaFlagY    = $5d  // 1 byte
    .const TempZP           = $5e  // 1 byte
    .const SimultaneousKeys = $5f  // 1 byte
```

```
    // Operational Variables
    .var MaxKeyRollover = 3

Keyboard:
{
    jmp Main


    //~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
    // Routine for Scanning a Matrix Row

KeyInRow:
    asl
    bcs *+5
        jsr KeyFound
    .for (var i = 0 ; i < 7 ; i++) {
        inx
        asl
        bcs *+5
            jsr KeyFound
    }
    rts


    //~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
    // Routine for handling: Key Found

KeyFound:
    stx TempZP
    dec KeyQuantity
    bmi OverFlow
    ldy KeyTable,x
    ldx KeyQuantity
    sty BufferNew,x
    ldx TempZP
    rts

    //~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
    // Routine for handling: Overflow

OverFlow:
    pla  // Dirty hack to handle 2 layers of JSR
    pla
    pla
    pla
    // Don't manipulate last legal buffer as the routine will fix itself
once it gets valid input again.
    lda #$03
    sec
    rts
```

```
    //~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
    // Exit Routine for: No Activity

NoActivityDetected:
    // Exit With A = #$01, Carry Set & Reset BufferOld.
    lda #$00
    sta SimultaneousAlphanumericKeysFlag  // Clear the too many keys flag
once a "no activity" state is detected.
    stx BufferOld
    stx BufferOld+1
    stx BufferOld+2
    sec
    lda #$01
    rts


    //~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
    // Exit Routine for Control Port Activity

ControlPort:
    // Exit with A = #$02, Carry Set. Keep BufferOld to verify input after
Control Port activity ceases
    sec
    lda #$02
    rts


    //~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
    // Configure Data Direction Registers
Main:
    ldx #$ff
    stx $dc02       // Port A - Output
    ldy #$00
    sty $dc03       // Port B - Input


    //~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
    // Check for Port Activity

    sty $dc00       // Connect all Keyboard Rows
    cpx $dc01
    beq NoActivityDetected

    lda SimultaneousAlphanumericKeysFlag
    beq !+
        // Waiting for all keys to be released before accepting new input.
        lda #$05
        sec
        rts
!:
```

```
    //~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
    // Check for Control Port #1 Activity

    stx $dc00       // Disconnect all Keyboard Rows
    cpx $dc01       // Only Control Port activity will be detected
    bne ControlPort


    //~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
    // Scan Keyboard Matrix

    lda #%11111110
    sta $dc00
    ldy $dc01
    sty ScanResult+7
    sec
    .for (var i = 6 ; i > -1 ; i--) {
        rol
        sta $dc00
        ldy $dc01
        sty ScanResult+i
    }


    //~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
    // Check for Control Port #1 Activity (again)

    stx $dc00       // Disconnect all Keyboard Rows
    cpx $dc01       // Only Control Port activity will be detected
    bne ControlPort


    //~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
    // Initialize Buffer, Flags and Max Keys

    // Reset current read buffer
    stx BufferNew
    stx BufferNew+1
    stx BufferNew+2

    // Reset Non-AlphaNumeric Flag
    inx
    stx NonAlphaFlagY

    // Set max keys allowed before ignoring result
    lda #MaxKeyRollover
    sta KeyQuantity

    // Counter to check for simultaneous alphanumeric key-presses
    lda #$fe
    sta SimultaneousKeys
```

```
    //~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
    // Check and flag Non Alphanumeric Keys

    lda ScanResult+6
    eor #$ff
    and #%10000000     // Left Shift
    lsr
    sta NonAlphaFlagY
    lda ScanResult+0
    eor #$ff
    and #%10100100     // RUN STOP - C= - CTRL
    ora NonAlphaFlagY
    sta NonAlphaFlagY
    lda ScanResult+1
    eor #$ff
    and #%00011000     // Right SHIFT - CLR HOME
    ora NonAlphaFlagY
    sta NonAlphaFlagY

    lda ScanResult+7  // The rest
    eor #$ff
    sta NonAlphaFlagX


    //~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
    // Check for pressed key(s)

    lda ScanResult+7
    cmp #$ff
    beq *+5
        jsr KeyInRow
    .for (var i = 6 ; i > -1 ; i--) {
        ldx #[7-i]*8
        lda ScanResult+i
        beq *+5
            jsr KeyInRow
    }


    //~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
    // Key Scan Completed

    // Put any new key (not in old scan) into buffer
    ldx #MaxKeyRollover-1
    !:  lda BufferNew,x
        cmp #$ff
        beq Exist        // Handle 'null' values
        cmp BufferOld
        beq Exist
        cmp BufferOld+1
```

```
        beq Exist
        cmp BufferOld+2
        beq Exist
            // New Key Detected
            inc BufferQuantity
            ldy BufferQuantity
            sta Buffer,y
            // Keep track of how many new Alphanumeric keys are detected
            inc SimultaneousKeys
            beq TooManyNewKeys
    Exist:
        dex
        bpl !-

    // Anything in Buffer?
    ldy BufferQuantity
    bmi BufferEmpty
        // Yes: Then return it and tidy up the buffer
        dec BufferQuantity
        lda Buffer
        ldx Buffer+1
        stx Buffer
        ldx Buffer+2
        stx Buffer+1
        jmp Return

BufferEmpty:  // No new Alphanumeric keys to handle.
    lda #$ff

Return:  // A is preset
    clc
    // Copy BufferNew to BufferOld
    ldx BufferNew
    stx BufferOld
    ldx BufferNew+1
    stx BufferOld+1
    ldx BufferNew+2
    stx BufferOld+2
    // Handle Non Alphanumeric Keys
    ldx NonAlphaFlagX
    ldy NonAlphaFlagY
    rts

TooManyNewKeys:
    sec
    lda #$ff
    sta BufferQuantity
    sta SimultaneousAlphanumericKeysFlag
    lda #$04
    rts
```

```
//~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
KeyTable:
    .byte $ff, $ff, $ff, $ff, $ff, $ff, $ff, $ff  // CRSR DOWN, F5, F3, F1,
F7, CRSR RIGHT, RETURN, INST DEL
    .byte $ff, $05, $13, $1a, $34, $01, $17, $33  // LEFT SHIFT, "E", "S",
"Z", "4", "A", "W", "3"
    .byte $18, $14, $06, $03, $36, $04, $12, $35  // "X", "T", "F", "C",
"6", "D", "R", "5"
    .byte $16, $15, $08, $02, $38, $07, $19, $37  // "V", "U", "H", "B",
"8", "G", "Y", "7"
    .byte $0e, $0f, $0b, $0d, $30, $0a, $09, $39  // "N", "O" (Oscar), "K",
"M", "0" (Zero), "J", "I", "9"
    .byte $2c, $00, $3a, $2e, $2d, $0c, $10, $2b  // ",", "@", ":", ".", "-
", "L", "P", "+"
    .byte $2f, $1e, $3d, $ff, $ff, $3b, $2a, $1c  // "/", "^", "=", RIGHT
SHIFT, HOME, ";", "*", "£"
    .byte $ff, $11, $ff, $20, $32, $ff, $1f, $31  // RUN STOP, "Q", "C="
(CMD), " " (SPC), "2", "CTRL", "<-", "1"

BufferOld:
    .byte $ff, $ff, $ff

Buffer:
    .byte $ff, $ff, $ff, $ff

BufferQuantity:
    .byte $ff

SimultaneousAlphanumericKeysFlag:
    .byte $00
}
```

That's it!

The test program: Keyboard IO Routine V2.5 @ CSDb