Interfacing the PC

The AT keyboard.

IBM Keyboards. Not really an interesting topic, one would expect. So why would you want to interface the Keyboard? The IBM keyboard can be a cheap alternative to a keyboard on a Microprocessor development system. Or maybe you want a remote terminal, just couple it with a LCD Module.

Maybe you have a RS-232 Barcode Scanner or other input devices, which you want to use with existing software which only allows you to key in numbers or letters. You could design yourself a little box to convert RS-232 into a Keyboard Transmission, making it transparent to the software.

An interfacing example is given showing the keyboard's protocols in action. This interfacing example uses a 68HC705J1A MCU to decode an IBM AT keyboard and output the ASCII equivalent of the key pressed at 9600 BPS.

Note that this page only deals with AT Keyboards. If you have any XT keyboards, you wish to interface, consider placing them in a museum. We will not deal with this type of keyboard in this document. XT Keyboards use a different protocol compared to the AT, thus code contained on this page will be incompatible.

PC Keyboard Theory

The IBM keyboard you most probably have sitting in front of you, sends scan codes to your computer. The scan codes tell your Keyboard Bios, what keys you have pressed or released. Take for example the 'A' Key. The 'A' key has a scan code of 1C (hex). When you press the 'A' key, your keyboard will send 1C down it's serial line. If you are still holding it down, for longer than it's typematic delay, another 1C will be sent. This keeps occurring until another key has been pressed, or if the 'A' key has been released.

However your keyboard will also send another code when the key has been released. Take the example of the 'A' key again, when released, the keyboard will send F0 (hex) to tell you that the key with the proceeding scan code has been released. It will then send 1C, so you know which key has been released.

Your keyboard only has one code for each key. It doesn't care it the shift key has been pressed. It will still send you the same code. It's up to your keyboard BIOS to determine this and take the appropriate action. Your keyboard doesn't even process the Num Lock, Caps Lock and Scroll Lock. When you press the Caps Lock for example, the keyboard will send the scan code for the cap locks. It is then up to your keyboard BIOS to send a code to the keyboard to turn on the Caps lock LED.

Now there's 101 keys and 8 bits make 256 different combinations, thus you only need to send one byte per key, right?

Nop. Unfortunately a handful of the keys found on your keyboard are extended keys, and thus require two scan code. These keys are preceded by a E0 (hex). But it doesn't stop at two scan codes either. How about E1,14,77,E1,F0,14,F0,77! Now that can't be a valid scan code? Wrong again. It's happens to be sent when you press the Pause/break key. Don't ask me why they have to make it so long! Maybe they were having a bad day or something?

When an extended key has been released, it would be expect that F0 would be sent to tell you that a key has been released. Then you would expect E0, telling you it was an extended key followed by the scan code for the key pressed. However this is not the case. E0 is sent first, followed by F0, when an extended key has been released.

Keyboard Commands

Besides Scan codes, commands can also be sent to and from the keyboard. The following section details the function of these commands. By no means is this a complete list. These are only some of the more common commands.

Host Commands

These commands are sent by the Host to the Keyboard. The most common command would be the setting/resetting of the Status Indicators (i.e. the Num lock, Caps Lock & Scroll Lock LEDs). The more common and useful commands are shown below.

ED Set Status LED's - This command can be used to turn on and off the Num Lock, Caps Lock & Scroll Lock LED's. After Sending ED, keyboard will reply with ACK (FA) and wait for another byte which determines their Status. Bit 0 controls the Scroll Lock, Bit 1 the Num Lock and Bit 2 the Caps lock. Bits 3 to 7 are ignored.

EE Echo - Upon sending a Echo command to the Keyboard, the keyboard should reply with a Echo (EE)

F0 Set Scan Code Set. Upon Sending F0, keyboard will reply with ACK (FA) and wait for another byte, 01-03 which determines the Scan Code Used. Sending 00 as the second byte will return the Scan Code Set currently in Use

F3 Set Typematic Repeat Rate. Keyboard will Acknowledge command with FA and wait for second byte, which determines the Typematic Repeat Rate.

F4 Keyboard Enable - Clears the keyboards output buffer, enables Keyboard Scanning and returns an Acknowledgment.

F5 Keyboard Disable - Resets the keyboard, disables Keyboard Scanning and returns an Acknowledgment.

FE Resend - Upon receipt of the resend command the keyboard will re-transmit the last byte sent.

FF Reset - Resets the Keyboard.

Commands

Now if the Host Commands are send from the host to the keyboard, then the keyboard commands must be sent from the keyboard to host. If you think this way, you must be correct. Below details some of the commands which the keyboard can send.

FA Acknowledge

AA Power On Self Test Passed (BAT Completed)

EE See Echo Command (Host Commands)

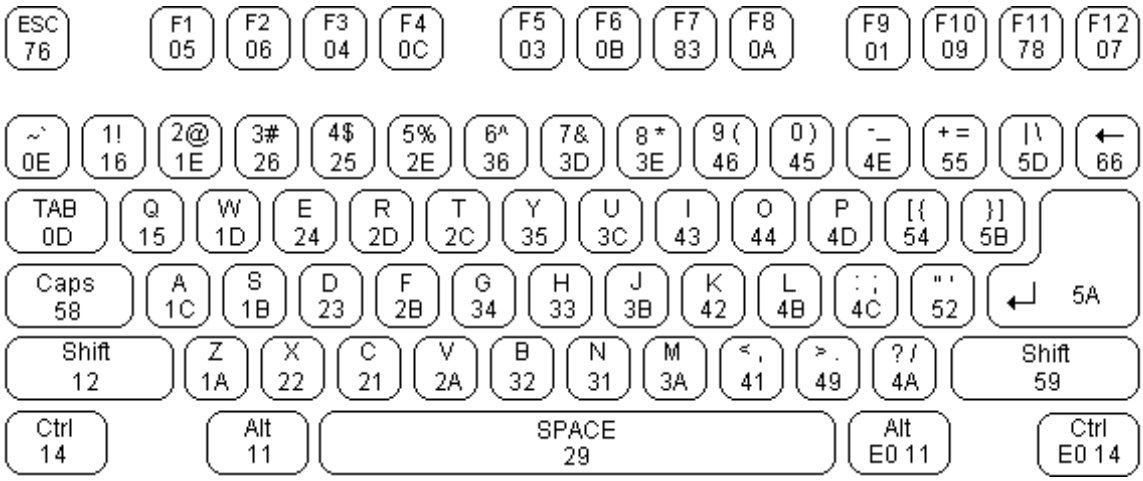
FE Resend - Upon receipt of the resend command the Host should re-transmit the last byte sent.

00 Error or Buffer Overflow

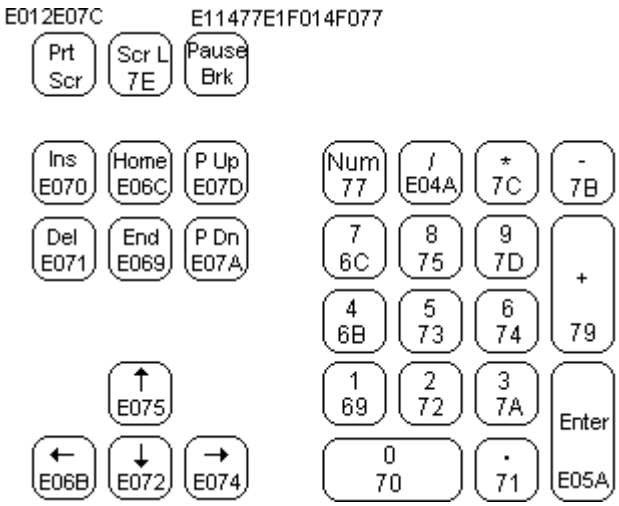
FF Error or Buffer Overflow

Scan Codes

The diagram below shows the Scan Code assigned to the individual keys. The Scan code is shown on the bottom of the key. E.g. The Scan Code for ESC is 76. All the scan codes are shown in Hex.



As you can see, the scan code assignments are quite random. In many cases the easiest way to convert the scan code to ASCII would be to use a look up table. Below is the scan codes for the extended keyboard & Numeric keypad.



The Keyboard's Connector

The PC's AT Keyboard is connected to external equipment using four wires. These wires are shown below for the Male Plug.

1. KBD Clock
2. KBD Data
3. N/C
4. GND
5. +5V (VCC)

A fifth wire can sometimes be found. This was once upon a time implemented as a Keyboard Reset, but today is left disconnected on AT Keyboards. Both the KBD Clock and KBD Data are Open Collector bi-directional I/O Lines. If desired, the Host can talk to the keyboard using these lines.

Note: Most keyboards are specified to drain a maximum 300mA. This will need to be considered when powering your devices

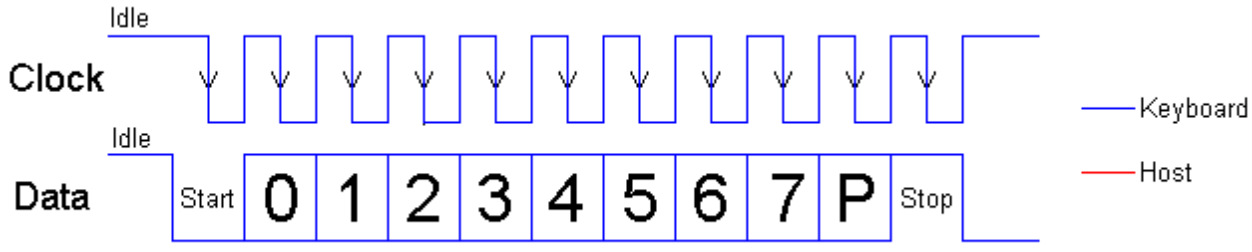
The Keyboard's Protocol

Keyboard to Host

As mentioned before, the PC's keyboard implements a bi-directional protocol. The keyboard can send data to the Host and the Host can send data to the Keyboard. The Host has the ultimate priority over direction. It can at anytime (although the not recommended) send a command to the keyboard.

The keyboard is free to send data to the host when both the KBD Data and KBD Clock lines are high (Idle). The KBD Clock line can be used as a Clear to Send line. If the host takes the KBD Clock line low, the keyboard will buffer any data until the KBD Clock is released, ie goes high. Should the Host take the KBD Data line low, then the keyboard will prepare to accept a command from the host.

The transmission of data in the forward direction, ie Keyboard to Host is done with a frame of 11 bits. The first bit is a Start Bit (Logic 0) followed by 8 data bits (LSB First), one Parity Bit (Odd Parity) and a Stop Bit (Logic 1). Each bit should be read on the falling edge of the clock.

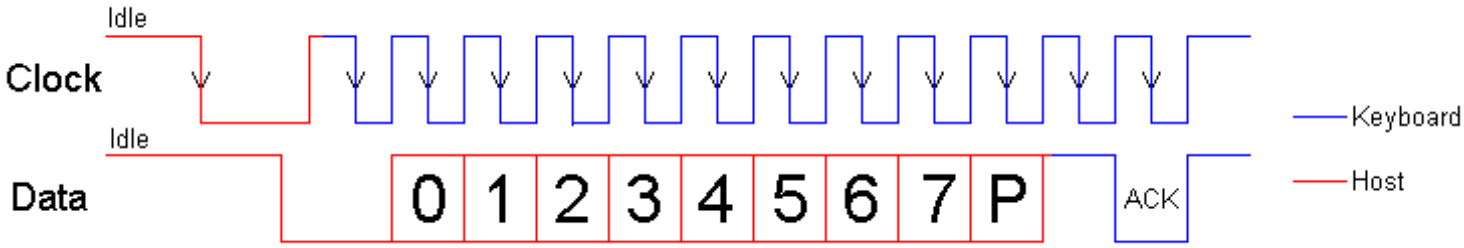


The above waveform represents a one byte transmission from the Keyboard. The keyboard may not generally change it's data line on the rising edge of the clock as shown in the diagram. The data line only has to be valid on the falling edge of the clock. The Keyboard will generate the clock. The frequency of the clock signal typically ranges from 20 to 30 Khz. The Least Significant Bit is always sent first.

Host to Keyboard

The Host to Keyboard Protocol is initiated by taking the KBD data line low. However to prevent the keyboard from sending data at the same time that you attempt to send the keyboard data, it is common to take the KBD Clock line low for more than 60us. This is more than one bit length. Then the KBD data line is taken low, while the KBD clock line is released.

The keyboard will start generating a clock signal on it's KBD clock line. This process can take up to 10mS. After the first falling edge has been detected, you can load the first data bit on the KBD Data line. This bit will be read into the keyboard on the next falling edge, after which you can place the next bit of data. This process is repeated for the 8 data bits. After the data bits come an Odd Parity Bit.



Once the Parity Bit has been sent and the KBD Data Line is in a idle (High) state for the next clock cycle, the keyboard will acknowledge the reception of the new data. The keyboard does this by taking the KBD Data line low for the next clock transition. If the KBD Data line is not idle after the 10th bit (Start, 8 Data bits + Parity), the keyboard will continue to send a KBD Clock signal until the KBD Data line becomes idle.

Interfacing Example - Keyboard to ASCII Decoder

Normally in this series of web pages, we connect something to the PC, to demonstrate the protocols at work. However this poses a problem with the keyboard. What could be possibly want to send to the computer via the keyboard interface?

Straight away any devious minds would be going, why not a little box, which generates passwords!. It could keep sending characters to the computer until it finds the right sequence. Well I'm not going to encourage what could possibly be illegal practices.

In fact a reasonably useful example will be given using a 68HC705J1A single chip microcontroller. We will get it to read the data from the keyboard, convert the scan codes into ASCII and send it out in RS-232 format at 9600 BPS. However we won't stop here, you will want to see the bi-directional use of the KBD Clock & Data lines, thus we will use the keyboards status LEDS, Num Lock, Caps Lock and Scroll Lock.

This can be used for quite a wide range of things. Teamed up with a reasonably sized 4 line x 40 character LCD panel, you could have yourself a little portable terminal. Or you could use it with a microcontroller development system. The 68HC705J1A in a One Time Programmable (OTP) is only a fraction of the cost of a 74C922 keyboard decoder chip, which only decodes a 4 x 4 matrix keypad to binary.

The keyboard doesn't need to be expensive either. Most people have many old keyboards floating around the place. If it's an AT Keyboard, then use it (XT keyboards will not work with this program.) If we ever see the introduction of USB keyboards, then there could be many redundant AT keyboards just waiting for you to hook them up.

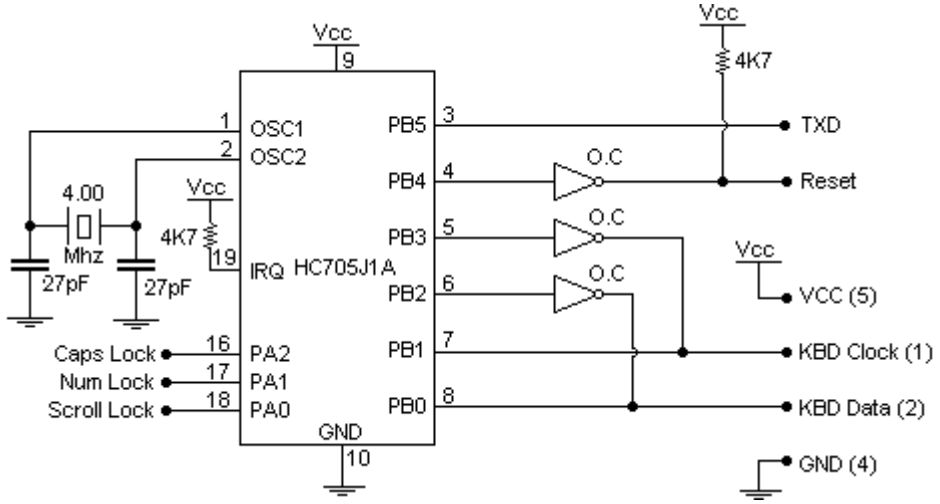
Features

Before we start with the technical aspects of the project, the salesman in me wants to tell you about the features packed into the 998 bytes of code.

- Use of the keyboard's bi-directional protocol allowing the status of the Num Lock, Caps Lock and Scroll Lock to be displayed on the Keyboards LEDs.
- External Reset Line activated by ALT-CTRL-DEL. If you are using it with a Microcontroler development system, you can reset the MCU with the keyboard. *I've always wanted to be able to use the three fingered solute on the HC11!*
- Scroll Lock and Num Lock toggles two Parallel Port Pins on the HC705. This can be used to turn things on or off, Select Memory Pages, Operating Systems etc
- "ALTDEC" or what I call the Direct Decimal Enter Routine. Just like using a PC, when you enter a decimal number when holding down one of the ALT keys the number is sent as binary to the target system. E.g. If you press and hold down ALT, then type in 255 and release ALT, the value FF (Hex) will be sent to the system. *Note. Unlike the PC, you can use both the numeric keypad or the numbers along the top of the keyboard.*
- "CTRLHEX" or you guessed it, Direct Hexadecimal Enter Routine. This function is not found with the PC. If you hold CTRL down, you can enter a Hexadecimal number. Just the thing for Development Systems or even debugging RS-232 Comms?
- Output is in ASCII using a RS-232 format at 9600 BPS. If using it with a development System, you can tap it in after the RS-232 Line Transceivers to save you a few dollars on RS-232 Level Converters.

Schematic & Hardware

The schematic below, shows the general connections of the keyboard to the HC705. The O.C. on the inverters denotes Open Collector outputs. I've used 74LS05 for the Open Collector Inverters, but it is up to you how you want to implement it. You can also use transistors with a suitable current limiting resistor, if you see fit.



The TXD pin, while it transmits in RS-232 format, is not at RS-232 Voltage Levels. If you want to connect it to RS-232 Devices then you will need to attach a RS-232 Level Converter of some kind. If you are using it with a development system, you can bypass both RS-232 Level Converters and connect it directly to the RXD pin of the MCU. However the keyboard can't be a direct replacement for a terminal on a development system, unless you want to type in your code each time! You may want to place a jumper or switch inline to switch between your RS-232 Port and the keyboard.

The circuit is designed to run on a 4Mhz crystal (2Mhz Bus Speed). The timing for the RS-232 transmission is based on the bus speed, thus this crystal has to be 4 Mhz. If you are lucky enough to have a 4 Mhz E Clock on your development system you can use it. I might at a later date try to create a 2 Mhz version which will run happily off a HC11 with a 8 Meg Crystal. This will reduce the cost of the project even further.

The power supply can also create a slight problem. A standard keyboard can drain about 300mA max, thus it would be recommended to use it's own regulator rather than taking a supply from elsewhere. Filter Capacitors are not shown on the general schematic but are implied for reliable operation. Consult your MC68HC705J1A Technical Data Manual for more information.

Reading Bytes from the Keyboard.

Now it is time to look at the code. I cannot include a description of all the code in this web page. The list file is just on 16 pages. Most of it (hopefully) is easy to follow. *Just like other code, count the number of spelling errors, while you are at it.*

```
Receive  ldx     #08                    ;Number of Bits
         clra                           ;Clear Parity Register
         bclr    clkout,PORTB           ;Clear to Send

         brset   clkin,PORTB,*          ;wait for idle Clock
         brset   datain,PORTB,Receive   ;False Start Bit, Restart
```

Remember the KBD Clock line? If you take it low, the keyboard will buffer any keys pressed. The Keyboard will only attempt to send when both the Data and Clock lines are idle (high). As it can take considerable time to decode the keys pressed, we must stop the keyboard from sending data. If not, some of the data may be lost or corrupted.

The program, will keep the KBD Clock line low, unless it is ready to accept data. We will use a loop to retrieve the data bits from the keyboard, thus we will load index register X with the number of bits be want to receive. The Accumulator will be used to verify the parity bit. We must clear this first.

We then place the KBD Clock line in the idle state so that the keyboard will start transmitting data if a key has been pressed. The program then loops while the Clock Line is Idle. One the KBD clock goes low, the loop is broken and the KBD Data Line is read. This should be the start bit which should be low. If not we branch to the start of the receive routine.

```
Recdata  ror     byte
         jsr     highlow                ;Wait for high to low Transition
         brset   datain,PORTB,Recset
         bclr    7,byte
         jmp     Recnext
Recset   bset    7,byte
         inca
Recnext  decx
         bne     Recdata                ;Loop until 8 bits been received
```

Once the Start bit has been detected, the 8 data bits must follow. The data is only valid on the falling edge of the clock. The subroutine *highlow* shown below will wait for the falling edge of the clock.

```
highlow brclr   clkin,PORTB,*         ;Loop until Clk High
        brset   clkin,PORTB,*         ;Loop until Clk Low
        rts
```

After the falling edge we read the level of the KBD Data line. If it is high we set the MSB of *byte* or if it is clear, we clear it. You will notice if the bit is set, we also increment the accumulator. This keeps track of the number of 1's in the byte and thus can be used to verify the Parity Bit. Index register X is decremented as we have read a byte. It then repeats the above process, until the entire 8 bits have been read.

```
        jsr     highlow
        eor     PORTB                 ;Parity Bit Detection
        and     #$01
        beq     r_error
```

After the 8 data bits, comes the dreaded Parity Bit. We could ignore it if we wanted to, but we may as well do something about it. We have been keeping a tally of the number of 1's in the Accumulator. The keyboard uses Odd parity, thus the parity bit should be the complement of the LSBit in the Accumulator. By exclusive OR-ing the Accumulator with the Parity Bit, we get a 1 if both the bits are different. I.e a '1' if the parity bit checks out.

As we are only interested in the LSB we can quite happy XOR the accumulator with PORTB. However this means the KBD datain must be connected to PB0. This can be a slight catch if you alter the equates. Then we single out the LSB using the AND function. If the resultant is zero, then a parity error has occurred and the program branches to r_error.

```
        jsr     highlow
        brclr   datain,PORTB,r_error    ;Stop Bit Detection

        bset    clkout,PORTB            ;Prevent Keyboard from sending data
                                        ;(Clear to Send)
        rts
```

After the Parity Bits comes the Stop Bit. Once again we can ignore it if we desire. However we have chosen to branch to an error routine if this occurs. The Stop bits should be set, thus an error occurs when it is clear.

```
r_error lda     #$FE                  ;Resend
        sta     byte
        jsr     Transmit
        jmp     Receive               ;Try again
```

What you do as error handling is up to you. In most cases it will never be executed. In fact I don't yet know if the above error handling routine works. I need to program another HC705 to send a false parity bit. *I've tried it out in close proximity to the Washing Machine, but I really need a controlled source!*

When an error occurs in the Parity or Stop Bit we should assume that the rest of the byte could have errors as well. We could ignore the error and process the received byte, but it could have unexpected results. Instead the keyboard has a resend command. If we issue a resend (FE) to the keyboard, the keyboard should send the byte back again. This is what occurs here.

You may notice that we branch to the error routine which transmits a resend command straight

away, without waiting for the corrupt transmission to finish. This is not a problem, as the keyboard considers any transmission to be successful, if the 10th bit is sent, i.e. the parity bit. If we interrupt the transmission before the parity bit is sent, the keyboard will place the current byte in it's buffer for later transmission.

You may of noticed that reading a byte doesn't really require bi-directional data and clock lines. If you can process the byte fast enough then no handshaking (RTS) is required. This means you no longer need the Open Collector inverters and the 2 Parallel Port lines. I have successfully done this with the HC705, outputting only scan codes on a Parallel Bus. But as you can imagine, you must be quick in order to catch the next transmission.

Writing Bytes to the Keyboard.

Writing commands to the keyboard involves the use of the Open Collector inverters. If we require an idle line (+5v), then we must transmit a 0 (zero) to achieve this. Everything which is sent via the dataout and clockout lines must be inverted.

```
transmit ldx     #$08                    ;8 Data Bits
         bset    clkout,PORTB            ;Set Clock Low
         lda     #$13                    ;Delay 64uS
         jsr     delay
         clra                            ;Clear Parity Register
         bset    dataout,PORTB           ;Set Data Low
         bclr    clkout,PORTB            ;Release Clock Line
         jsr     highlow
```

The routine given here is a generic one which can be used for your own purposes. During normal execution of this program the KBD clock line should be low, to prevent data being sent when the MCU isn't ready for it. However in this example, we take low the KBD clock line and wait for the 64uS which is pointless as the line is already low and has been like this for quite some time, since the end of the last Transmission or Reception.

The program then initiates the Host to Keyboard transmission by taking the KBD data line low and releasing the KBD clock line. We must then wait for a high to low transition on the KBD clock, before we load the first bit on the KBD data line. - *Something which is not clear in other FAQ's on the net.*

```
loop     ror     byte
         bcs     mark
space    bset    dataout,PORTB           ; Clear Bit
         jmp     next
mark     bclr    dataout,PORTB           ; Clear Bit
         inca                            ; Parity Calculation
next     jsr     highlow                 ; Wait for high to low transition
         decx
         bne     loop
```

The loading of the individual bits on the KBD data line is done in very similar fashion to the read cycle. However note that the bits are inverted. Also like the read cycle, we increment the accumulator so we can calculate the parity bit later on.

```
         and     #$01
         bne     clr_par
set_par  bclr    dataout,PORTB
         jmp     tr_ackn
clr_par  bset    dataout,PORTB
tr_ackn  jsr     highlow
```

After the data bits have been sent, it is now time to send the parity bit. Unlike the read cycle, we can't ignore the parity bit. If we do the keyboard will issue a resend (FE) command if the parity bit is incorrect.

```
          bclr    dataout,PORTB           ;Release Data Line
          jsr     highlow
          brset   datain,PORTB,error      ;Check for Ack
          brclr   clkin,PORTB,*           ;Wait for idle line

          bset    clkout,PORTB            ;Prevent Keyboard from sending data
                                          ;(Clear to Send)
          rts
```

Once the Parity bit has been set and the falling edge of the KBD clock detected, we must release the KBD data line, wait for another falling edge of the KBD clock to see if the Keyboard has acknowledged the byte. The keyboard does this by pulling the KBD data line low. If it is not low, then the program branches to an error handler. If all has been successful, the MCU pulls down the KBD clock, to prevent it from transmitting.

```
error     lda     #$FF      ;Reset
          sta     byte
          jsr     transmit
          rts
```

We have taken a harsher approach to handing any transmit errors. Ideally we should wait for the keyboard to send a resend command and then retransmit the byte. However what we have done is to issue a reset of the keyboard. So far I've never had an error, however if this starts to become a problem, then a better error handler will be written.

Source Code

```
KEYBRD05.ASM            Assembled with CASM   02/15/1998  22:12  PAGE 1
                  1  ****************************************************
                  2  *                                                  *
                  3  *   101 Key, IBM Keyboard Decoder for 68HC705J1A.   *
                  4  *                                                  *
                  5  *     Copyright 1997 / 1998 - Craig Peacock        *
                  6  *                15th February 1998                *
                  7  *                                                  *
                  8  *        Includes ALTDEC & CTRLHEX Routines        *
                  9  *                                                  *
                 10  ****************************************************
                 11
 0300            12  datain  equ     0       ; Must be LSB - See Parity Calculations
 0300            13  clkin   equ     1
 0300            14  dataout equ     2
 0300            15  clkout  equ     3
 0300            16  nreset  equ     4       ; Reset Output
 0300            17  TXD     equ     5       ; Transmit Pin on Port B
                 18
                 19  ; Equates for LED Byte
                 20
 0300            21  pscrlck equ     7       ; If true, Scroll Lock Pressed
 0300            22  pnumlck equ     6       ; If true, Num Lock Pressed
                 23
 0300            24  caplock equ     2       ; If true, Caps Lock is On (Active)
 0300            25  numlock equ     1       ; If true, Num Lock is On (Active)
 0300            26  scrlock equ     0       ; If true, Scroll Lock is On (Active)
                 27
                 28  ; Equates for Status Flag, Byte
                 29
 0300            30  rctrl   equ     7       ; If true, Right Ctrl Pressed
 0300            31  lctrl   equ     6       ; If true, Left Ctrl Pressed
 0300            32  ralt    equ     5       ; If true, Right Alt Pressed
 0300            33  lalt    equ     4       ; If true, Left Alt Pressed
                 34
 0300            35  caploc  equ     2       ; If true, Caps Lock Pressed
 0300            36  rshift  equ     1       ; If true, Right Shift Key Pressed
 0300            37  lshift  equ     0       ; If true, Left Shift Key Pressed
                 38
 00C0            39          org     ram
                 40
 00C0            41  byte    rmb     1       ; Used to hold byte, during Trans & Rec
 00C1            42  status  rmb     1       ; Status Flags
 00C2            43  LED     rmb     1       ; LED Flags
 00C3            44  asc     rmb     3       ; Used for altdec & ctrlhex
                 45
 07F8            46          org     $7F8
                 47
 07F8 0300       48          dw      start   ; Timer Interrupt Vector
 07FA 0300       49          dw      start   ; IRQ Vector
 07FC 0300       50          dw      start   ; Software Interrupt Vector
 07FE 0300       51          dw      start   ; Reset Vector
                 52
 0300            53          org     rom
                 54
 0300 A6FF       55  start   lda     #%11111111      ;PORTA
 0302 B704       56          sta     ddra            ;Set Data Direction Register
 0304 B710       57          sta     pdra            ;Enable Pull Downs
                 58
```

```
0306 A6FC        59          lda     #%11111100      ;PORTB
0308 B705        60          sta     ddrb            ;Set Data Direction Register
030A B711        61          sta     pdrb            ;Enable Pull Downs
                 62
030C 1A01        63          bset    TXD,PORTB       ;Transmit Line Idle
030E 1501        64          bclr    dataout,PORTB   ;KBD Data Idle
0310 1701        65          bclr    clkout,PORTB    ;KBD Clock Idle
0312 1901        66          bclr    nreset,PORTB    ;Reset Line Idle
                 67
0314 CC031E      68          jmp     rstflag         ;No Attempt to Reset Keyboard made
                 69                                  ;as keyboard would still be in POST
                 70                                  ;or BAT Tests, if power applied at
                 71                                  ;the same time than the HC705.
                 72
                 73  ************************************************
                 74  *                                              *
                 75  * reset - Sends a Reset Command to the Keyboard.    *
                 76  *         Not a very good effort to reset keyboard, *
                 77  *         as it doesn't check for ACK or BAT        *
                 78  *         Completion Code. I.e. Reset may not of    *
                 79  *         even Worked!                              *
                 80  *                                              *
                 81  ************************************************
                 82
0317 A6FF        83  reset   lda     #$FF            ;Reset Keyboard
0319 B7C0        84          sta     byte
031B CD0497      85          jsr     transmit
                 86
                 87  ************************************************
                 88  *                                              *
                 89  * rstflag - Resets Status and LED Flags. Used when  *
                 90  *           a successful Bat Completion code is     *
                 91  *           sent to sync keyboard's LED's to 705's  *
                 92  *           status register                         *
                 93  *                                              *
                 94  ************************************************
                 95
031E 3FC1        96  rstflag clr     status
0320 3FC2        97          clr     LED
                 98
                 99  ************************************************
                100  *                                              *
                101  * main - Main Keyboard Decoding Routine. Once key   *
                102  *        been decoded, program should return here   *
                103  *                                              *
                104  ************************************************
                105
0322 CD04E2     106  main    jsr     Receive ;Get's a Single Byte from the Keyboard.
0325 B6C0       107          lda     byte
                108
0327 A1F0       109          cmp     #$F0    ;A Key has been Released
0329 2603       110          bne     main1
032B CC0429     111          jmp     release
                112
032E A1AA       113  main1   cmp     #$AA    ;Successful Completion of BAT
0330 2603       114          bne     main2
0332 CC031E     115          jmp     rstflag
                116
```

```
0335 A1E0       117  main2   cmp     #$E0    ;Extended Keys
0337 2603       118          bne     main3
0339 CC03D6     119          jmp     extend
                120
033C A112       121  main3   cmp     #$12    ;Left Shift Key Pressed
033E 2602       122          bne     main4
0340 10C1       123          bset    lshift,status
                124
0342 A159       125  main4   cmp     #$59    ;Right Shift Key Pressed
0344 2602       126          bne     main5
0346 12C1       127          bset    rshift,status
                128
0348 A114       129  main5   cmp     #$14    ;Left Ctrl
034A 2605       130          bne     main6
034C 1CC1       131          bset    lctrl,status
034E CC0588     132          jmp     clrasc
                133
0351 A111       134  main6   cmp     #$11    ;Left Alt
0353 2605       135          bne     main7
0355 18C1       136          bset    lalt,status
0357 CC0588     137          jmp     clrasc
                138
035A A158       139  main7   cmp     #$58    ;Caps Lock Pressed
035C 2605       140          bne     main8
035E 05C154     141          brclr   caploc,status,caps
0361 14C1       142          bset    caploc,status
                143
0363 A17E       144  main8   cmp     #$7E    ;Scroll Lock Pressed
0365 2605       145          bne     main9
0367 0FC161     146          brclr   pscrlck,status,scrl
036A 1EC1       147          bset    pscrlck,status
                148
036C A177       149  main9   cmp     #$77    ;Num Lock Pressed
036E 2605       150          bne     main10
0370 0DC14D     151          brclr   pnumlck,status,nums
0373 1CC1       152          bset    pnumlck,status
                153
0375 A18F       154  main10  cmp     #$8F    ;Last Value in Look-Up Table
0377 2503       155          blo     main11
0379 CC0322     156          jmp     main    ;Out of Bounds
                157
037C 97         158  main11  tax
037D 04C20C     159          brset   caplock,LED,caps_on
0380 02C10F     160          brset   rshift,status,shifton
0383 00C10C     161          brset   lshift,status,shifton
                162
0386 D605C6     163  cancel  lda     noshift,x               ;Load Lower Case Values
0389 CC0395     164          jmp     main12
                165
038C 02C1F7     166  caps_on brset   rshift,status,cancel    ;If ShiftLock & Shift, Cancel
038F 00C1F4     167          brset   lshift,status,cancel
                168
0392 D60656     169  shifton lda     shift,x                 ;Load Upper Case Values
                170
0395 271B       171  main12  beq     return                  ;Scan Code not in Lookup Table.
                172
0397 97         173          tax
0398 B6C1       174          lda     status
```

```
039A A430    175          and     #$30                    ;Either Alt Key Pressed
039C 2704    176          beq     main13
039E 9F      177          txa
039F CC053D  178          jmp     altdec
             179
03A2 B6C1    180  main13  lda     status
03A4 A4C0    181          and     #$C0                    ;Either CTRL Key Pressed
03A6 2704    182          beq     main14
03A8 9F      183          txa
03A9 CC0523  184          jmp     ctrlhex
             185
03AC 9F      186  main14  txa
03AD B7C0    187          sta     byte
03AF CD0591  188          jsr     RS232T                  ;Send to RS232
             189
03B2 CC0322  190  return  jmp     main
             191
             192  ***************************************************
             193  *                                                 *
             194  * caps - Toggle Status of Caps lock and Echo to   *
             195  *        Keyboard                                 *
             196  *                                                 *
             197  ***************************************************
             198
03B5 14C1    199  caps    bset    caploc,status   ; Set caploc flag to prevent routine being
             200                                   ; called again
03B7 B6C2    201          lda     LED
03B9 A804    202          eor     #$04            ; Toggle Shift Lock Flag
03BB B7C2    203          sta     LED
03BD CC047B  204          jmp     LEDshow
             205
             206  ***************************************************
             207  *                                                 *
             208  * nums - Toggle Status of Nums lock and Echo to   *
             209  *        Keyboard                                 *
             210  *                                                 *
             211  ***************************************************
             212
03C0 1CC1    213  nums    bset    pnumlck,status
             214
03C2 B6C2    215          lda     LED
03C4 A802    216          eor     #$02
03C6 B7C2    217          sta     LED
03C8 CC047B  218          jmp     LEDshow
             219
             220  ***************************************************
             221  *                                                 *
             222  * scrl - Toggle Status of Scroll lock and Echo to *
             223  *        Keyboard                                 *
             224  *                                                 *
             225  ***************************************************
             226
03CB 1EC1    227  scrl    bset    pscrlck,status
             228
03CD B6C2    229          lda     LED
03CF A801    230          eor     #$01
03D1 B7C2    231          sta     LED
03D3 CC047B  232          jmp     LEDshow
```

```
                    233
                    234  ***************************************************
                    235  *                                                 *
                    236  * extend - An Extended Key has been Pressed       *
                    237  *                                                 *
                    238  ***************************************************
                    239
 03D6 CD04E2        240  extend  jsr     Receive ;Get Next byte
 03D9 B6C0          241          lda     byte
                    242
 03DB A1F0          243          cmp     #$F0    ;An Extended Key Has been Released
 03DD 2603          244          bne     extend1
 03DF CC0461        245          jmp     rel_ext
                    246
 03E2 A111          247  extend1 cmp     #$11    ;Right Alt Pressed
 03E4 2605          248          bne     extend2
 03E6 1AC1          249          bset    ralt,status
 03E8 CC0588        250          jmp     clrasc
                    251
 03EB A114          252  extend2 cmp     #$14    ;Right Ctrl Pressed
 03ED 2605          253          bne     extend3
 03EF 1EC1          254          bset    rctrl,status
 03F1 CC0588        255          jmp     clrasc
                    256
 03F4 A171          257  extend3 cmp     #$71    ;Delete
 03F6 2618          258          bne     extend4
 03F8 B6C1          259          lda     status
 03FA A4C0          260          and     #$C0    ;Either Alt Key Pressed?
 03FC 2712          261          beq     extend4
 03FE B6C1          262          lda     status
 0400 A430          263          and     #$30    ;Either Ctrl Key Pressed?
 0402 270C          264          beq     extend4
 0404 1801          265          bset    nreset,PORTB
 0406 A6FF          266          lda     #$FF
 0408 CD05C2        267          jsr     delay
 040B 1901          268          bclr    nreset,PORTB
 040D CC0317        269          jmp     reset
                    270
 0410 A15A          271  extend4 cmp     #$5A    ;Enter Key on Num Keypad
 0412 2607          272          bne     extend5
 0414 A60D          273          lda     #$0D
 0416 B7C0          274          sta     byte
 0418 CD0591        275          jsr     RS232T
                    276
 041B A14A          277  extend5 cmp     #$4A    ; '/' Key on Num Keypad
 041D 2607          278          bne     extend6
 041F A62F          279          lda     #'/'
 0421 B7C0          280          sta     byte
 0423 CD0591        281          jsr     RS232T
                    282
 0426 CC0322        283  extend6 jmp     main    ;Return to main
                    284
                    285  ***************************************************
                    286  *                                                 *
                    287  * release - A Key has been Released               *
                    288  *                                                 *
                    289  ***************************************************
                    290
```

```
0429 CD04E2     291  release jsr     Receive ;Release - Next Byte Garbage in many cases
042C B6C0       292          lda     byte
                293
042E A112       294  releas3 cmp     #$12    ;Left Shift Key Released
0430 2602       295          bne     releas4
0432 11C1       296          bclr    lshift,status
                297
0434 A159       298  releas4 cmp     #$59    ;Right Shift Key Released
0436 2602       299          bne     releas5
0438 13C1       300          bclr    rshift,status
                301
043A A114       302  releas5 cmp     #$14    ;Left Ctrl Released
043C 2605       303          bne     releas6
043E 1DC1       304          bclr    lctrl,status
0440 CC0572     305          jmp     ctrl_re
                306
0443 A111       307  releas6 cmp     #$11    ;Left Alt Released
0445 2605       308          bne     releas7
0447 19C1       309          bclr    lalt,status
0449 CC0554     310          jmp     alt_rel
                311
044C A158       312  releas7 cmp     #$58     ;Caps Lock Released
044E 2602       313          bne     releas8
0450 15C1       314          bclr    caploc,status
                315
0452 A17E       316  releas8 cmp     #$7E     ;Scroll Lock Released
0454 2602       317          bne     releas9
0456 1FC1       318          bclr    pscrlck,status
                319
0458 A177       320  releas9 cmp     #$77     ;Num Lock Released
045A 2602       321          bne     relea10
045C 1DC1       322          bclr    pnumlck,status
                323
045E CC0322     324  relea10 jmp     main     ;Return to Main
                325
                326  ************************************************
                327  *                                              *
                328  * rel_ext - An Extended Key has been Released  *
                329  *                                              *
                330  ************************************************
                331
0461 CD04E2     332  rel_ext jsr     Receive  ;Get Next byte
0464 B6C0       333          lda     byte
                334
0466 A111       335          cmp     #$11    ;Right Alt Released
0468 2605       336          bne     rel_ex2
046A 1BC1       337          bclr    ralt,status
046C CC0554     338          jmp     alt_rel
                339
046F A114       340  rel_ex2 cmp     #$14    ;Right Ctrl Released
0471 2605       341          bne     rel_ex3
0473 1FC1       342          bclr    rctrl,status
0475 CC0572     343          jmp     ctrl_re
                344
0478 CC0322     345  rel_ex3 jmp     main    ;Return to main
                346
                347  ************************************************
                348  *                                              *
```

```
                349  * LEDshow - Copies the 3 LSB of the LED register to *
                350  *           keyboard for the keyboards Status LED's *
                351  *           E.g. Num Lock, Caps Lock, Scroll Lock   *
                352  *           Also makes their status present on      *
                353  *           PORTA                                   *
                354  *                                                   *
                355  *****************************************************
                356
 047B B6C2      357  LEDshow lda     LED
 047D A407      358          and     #$07
 047F B700      359          sta     PORTA   ;Made Status Avalible at PORTA
 0481 A6ED      360          lda     #$ED
 0483 B7C0      361          sta     byte
 0485 CD0497    362          jsr     transmit
 0488 CD04E2    363          jsr     Receive
 048B B6C2      364          lda     LED
 048D A407      365          and     #$07
 048F B7C0      366          sta     byte
 0491 CD0497    367          jsr     transmit
 0494 CC0322    368          jmp     main
                369
                370  *****************************************************
                371  *                                                   *
                372  * Transmit - Send Data stored at Byte to the        *
                373  *            Keyboard. Result                       *
                374  *                                                   *
                375  *****************************************************
                376
                377  transmit
                378
 0497 1701      379          bclr    clkout,PORTB
 0499 1501      380          bclr    dataout,PORTB          ;Make sure outputs are low.
                381
 049B AE08      382          ldx     #$08                   ;8 Data Bits
 049D 1601      383          bset    clkout,PORTB           ;Set Clock Low
 049F A613      384          lda     #$13                   ;Delay 64uS
 04A1 CD05C2    385          jsr     delay
 04A4 4F        386          clra                           ;Clear Parity Register
 04A5 1401      387          bset    dataout,PORTB          ;Set Data Low
 04A7 1701      388          bclr    clkout,PORTB           ;Release Clock Line
 04A9 CD051C    389          jsr     highlow
                390
                391
 04AC 36C0      392  loop    ror     byte
 04AE 2505      393          bcs     mark
                394
 04B0 1401      395  space   bset    dataout,PORTB          ; Clear Bit
 04B2 CC04B8    396          jmp     next
                397
 04B5 1501      398  mark    bclr    dataout,PORTB          ; Clear Bit
 04B7 4C        399          inca                           ; Parity Calculation
                400
 04B8 CD051C    401  next    jsr     highlow                ; Wait for high to low transition
                402
 04BB 5A        403          decx
 04BC 26EE      404          bne     loop
                405
 04BE A401      406          and     #$01
```

```
04C0 2605       407          bne     clr_par
04C2 1501       408  set_par bclr    dataout,PORTB
04C4 CC04C9     409          jmp     tr_ackn
04C7 1401       410  clr_par bset    dataout,PORTB
04C9 CD051C     411  tr_ackn jsr     highlow
                412
04CC 1501       413          bclr    dataout,PORTB          ;Release Data Line
04CE CD051C     414          jsr     highlow
04D1 000106     415          brset   datain,PORTB,error     ;Check for Ack
04D4 0301FD     416          brclr   clkin,PORTB,*          ;Wait for idle line
                417
04D7 1601       418          bset    clkout,PORTB           ;Prevent Keyboard from sending data
                419                                         ;(Clear to Send)
04D9 81         420          rts
                421
04DA A6FF       422  error   lda     #$FF      ;Reset
04DC B7C0       423          sta     byte
04DE CD0497     424          jsr     transmit
04E1 81         425          rts
                426
                427  ************************************************
                428  *                                              *
                429  * Receive - Get a Byte from the Keyboard. Result    *
                430  *           stored in byte.                    *
                431  *                                              *
                432  ************************************************
                433
04E2 AE08       434  Receive ldx     #08                    ;Number of Bits
04E4 4F         435          clra                           ;Clear Parity Register
04E5 1701       436          bclr    clkout,PORTB           ;Clear to Send
                437
04E7 0201FD     438          brset   clkin,PORTB,*          ;wait for idle Clock
04EA 0001F5     439          brset   datain,PORTB,Receive   ;False Start Bit, Restart
                440
04ED 36C0       441  Recdata ror     byte
04EF CD051C     442          jsr     highlow                ;Wait for high to low Transition
04F2 000105     443          brset   datain,PORTB,Recset
                444
04F5 1FC0       445          bclr    7,byte
04F7 CC04FD     446          jmp     Recnext
                447
04FA 1EC0       448  Recset  bset    7,byte
04FC 4C         449          inca
                450
04FD 5A         451  Recnext decx
04FE 26ED       452          bne     Recdata                ;Loop until 8 bits been received
                453
0500 CD051C     454          jsr     highlow
0503 B801       455          eor     PORTB                  ;Parity Bit Detection
0505 A401       456          and     #$01
0507 2709       457          beq     r_error
                458
0509 CD051C     459          jsr     highlow
050C 010103     460          brclr   datain,PORTB,r_error   ;Stop Bit Detection
                461
050F 1601       462          bset    clkout,PORTB           ;Prevent Keyboard from sending data
                463                                         ;(Clear to Send)
0511 81         464          rts
```

```
                465
0512 A6FE       466  r_error lda     #$FE                    ;Resend
0514 B7C0       467          sta     byte
0516 CD0497     468          jsr     Transmit
0519 CC04E2     469          jmp     Receive                 ;Try again
                470
                471  ************************************************
                472  *                                              *
                473  * highlow - Waits for next High to Low Transistion  *
                474  *           on the Clock Line                  *
                475  *                                              *
                476  ************************************************
                477
                478
051C 0301FD     479  highlow brclr   clkin,PORTB,*         ;Loop until Clk High
051F 0201FD     480          brset   clkin,PORTB,*         ;Loop until Clk Low
0522 81         481          rts
                482
                483  ************************************************
                484  *                                              *
                485  * ctrlhex & althex - Make sure keys pressed are     *
                486  * valid. If not, don't store them. Also converts    *
                487  * ASCII to binary and stores them in the ASCII      *
                488  * Storage Location                             *
                489  *                                              *
                490  ************************************************
                491
0523 A161       492  ctrlhex cmp     #'a'            ;Convert 'a' - 'f' to binary
0525 2509       493          blo     he1_ran
0527 A166       494          cmp     #'f'
0529 2226       495          bhi     outrang
052B A057       496          sub     #$57
052D CC0547     497          jmp     store
                498
0530 A141       499  he1_ran cmp     #'A'            ;Convert 'A' - 'F' to binary
0532 2509       500          blo     altdec
0534 A146       501          cmp     #'F'
0536 2219       502          bhi     outrang
0538 A037       503          sub     #$37
053A CC0547     504          jmp     store
                505
053D A130       506  altdec  cmp     #'0'            ;Convert '0' - '9' to binary
053F 2510       507          blo     outrang
0541 A139       508          cmp     #'9'
0543 220C       509          bhi     outrang
0545 A030       510          sub     #$30
                511
0547 BEC4       512  store   ldx     asc+1           ;Shift Bytes Left
0549 BFC3       513          stx     asc+0
054B BEC5       514          ldx     asc+2
054D BFC4       515          stx     asc+1
054F B7C5       516          sta     asc+2           ;Store as Binary
0551 CC0322     517  outrang jmp     main
                518
                519  ************************************************
                520  *                                              *
                521  * alt_rel Alt Released. (Decimal Enter Routine)     *
                522  *         Once both the ALT keys have been released *
```

```
                523  *         a calculation must be made to convert the *
                524  *         bytes found in ASCII Storage to binary    *
                525  *         for transmission.                         *
                526  *                                                   *
                527  *****************************************************
                528
 0554 B6C1      529  alt_rel lda     status          ;Decimal Calculation
 0556 A430      530          and     #$30
 0558 262B      531          bne     complet         ;One of the Alt Keys Still Pressed
                532
 055A B6C3      533          lda     asc
 055C AE64      534          ldx     #$64    ;x 100
 055E 42        535          mul
 055F B7C0      536          sta     byte
                537
 0561 B6C4      538          lda     asc+1
 0563 AE0A      539          ldx     #$0A    ;x 10
 0565 42        540          mul
 0566 BBC5      541          add     asc+2   ;Add Units
 0568 BBC0      542          add     byte    ;Add hundreds
 056A B7C0      543          sta     byte
                544
 056C CD0591    545          jsr     RS232T  ;Transmit number
                546
 056F CC0322    547          jmp     main    ;Return to Main.
                548
                549  *****************************************************
                550  *                                                   *
                551  * ctrl_re Ctrl Released.(Hexadecimal Enter Routine) *
                552  *         Once both the CTRL keys have been released*
                553  *         a calculation must be made to convert the *
                554  *         bytes found in ASCII Storage to binary    *
                555  *         for transmission.                         *
                556  *                                                   *
                557  *****************************************************
                558
 0572 B6C1      559  ctrl_re lda     status
 0574 A4C0      560          and     #$C0
 0576 260D      561          bne     complet         ;One of the Ctrl Keys Still Pressed
                562
 0578 B6C4      563          lda     asc+1
 057A 48        564          lsla
 057B 48        565          lsla
 057C 48        566          lsla
 057D 48        567          lsla
 057E BBC5      568          add     asc+2
 0580 B7C0      569          sta     byte
                570
 0582 CD0591    571          jsr     RS232T          ;Transmit Number
                572
 0585 CC0322    573  complet jmp     main            ;Return to Main
                574
                575  *****************************************************
                576  *                                                   *
                577  * clrasc - Clear ASCII Storage Locations (3 Bytes)  *
                578  *        - These storage bytes are used for the     *
                579  *          ALTDEC & CTRLHEX Routines.               *
                580  *                                                   *
```

```
                581  ***************************************************
                582
 0588 3FC3      583  clrasc  clr     asc+0
 058A 3FC4      584          clr     asc+1
 058C 3FC5      585          clr     asc+2
 058E CC0322    586          jmp     main
                587
                588  ***************************************************
                589  *                                                 *
                590  * RS-232 NRZ 8N1 Transmit Routine.                *
                591  *                                                 *
                592  * Uses a 4.00 Mhz Crystal (2 Mhz Bus Speed)       *
                593  * to obtain a transmission speed of 9600 BPS      *
                594  *                                                 *
                595  ***************************************************
                596
 0591 AE08      597  RS232T  ldx     #8        ; Number of Bits (8)
 0593 1B01      598          bclr    TXD,portb ; Start Bit (0)
 0595 A61D      599          lda     #$1D      ; 29 Cycles 6[29] + 6
 0597 CD05C2    600          jsr     delay
 059A 21FE      601          brn     *
 059C 9D        602          nop
 059D 9D        603          nop
 059E 36C0      604  data    ror     byte
 05A0 2505      605          bcs     rsmark
 05A2 1B01      606          bclr    TXD,portb ; Space (Logic 0)
 05A4 CC05AB    607          jmp     rsnext
 05A7 1A01      608  rsmark  bset    TXD,portb ; Mark (Logic 1)
 05A9 21FE      609          brn     *
 05AB A61C      610  rsnext  lda     #$1C      ; 28 Cycles 6[28] + 6
 05AD CD05C2    611          jsr     delay
 05B0 9D        612          nop
 05B1 9D        613          nop
 05B2 5A        614          decx
 05B3 26E9      615          bne     data
 05B5 21FE      616          brn     *
 05B7 21FE      617          brn     *
 05B9 9D        618          nop
 05BA 1A01      619          bset    TXD,portb ; Stop Bit (Logic 1)
 05BC A61F      620          lda     #$1F      ; 31 Cycles 6[31] + 6
 05BE CD05C2    621          jsr     delay
 05C1 81        622          rts
                623
 05C2 4A        624  delay   deca              ; Delay = 6[A] + 6
 05C3 26FD      625          bne     delay
 05C5 81        626          rts
                627
                628  ***************************************************
                629  *                                                 *
                630  *   No-Shift - Lookup Table when Shift not Pressed  *
                631  *                                                 *
                632  ***************************************************
                633
 05C6 00        634  noshift fcb      $00     ; 00
 05C7 00        635          fcb      $00     ; 01  F9
 05C8 00        636          fcb      $00     ; 02
 05C9 00        637          fcb      $00     ; 03  F5
 05CA 00        638          fcb      $00     ; 04  F3
```

```
05CB 00        639          fcb     $00     ; 05  F1
05CC 00        640          fcb     $00     ; 06  F2
05CD 00        641          fcb     $00     ; 07  F12
05CE 00        642          fcb     $00     ; 08
05CF 00        643          fcb     $00     ; 09  F10
05D0 00        644          fcb     $00     ; 0A  F8
05D1 00        645          fcb     $00     ; 0B  F6
05D2 00        646          fcb     $00     ; 0C  F4
05D3 09        647          fcb     $09     ; 0D  TAB
05D4 60        648          fcb     '`'     ; 0E  ` or ~
05D5 00        649          fcb     $00     ; 0F
               650
05D6 00        651          fcb     $00     ; 10
05D7 00        652          fcb     $00     ; 11  Left ALT
05D8 00        653          fcb     $00     ; 12  Left SHIFT
05D9 00        654          fcb     $00     ; 13
05DA 00        655          fcb     $00     ; 14  Left Ctrl
05DB 71        656          fcb     'q'     ; 15  Q
05DC 31        657          fcb     '1'     ; 16  1 or !
05DD 00        658          fcb     $00     ; 17
05DE 00        659          fcb     $00     ; 18
05DF 00        660          fcb     $00     ; 19
05E0 7A        661          fcb     'z'     ; 1A  Z
05E1 73        662          fcb     's'     ; 1B  S
05E2 61        663          fcb     'a'     ; 1C  A
05E3 77        664          fcb     'w'     ; 1D  W
05E4 32        665          fcb     '2'     ; 1E  2 or @
05E5 00        666          fcb     $00     ; 1F
               667
05E6 00        668          fcb     $00     ; 20
05E7 63        669          fcb     'c'     ; 21  C
05E8 78        670          fcb     'x'     ; 22  X
05E9 64        671          fcb     'd'     ; 23  D
05EA 65        672          fcb     'e'     ; 24  E
05EB 34        673          fcb     '4'     ; 25  4 or $
05EC 33        674          fcb     '3'     ; 26  3 or #
05ED 00        675          fcb     $00     ; 27
05EE 00        676          fcb     $00     ; 28
05EF 20        677          fcb     ' '     ; 29  Space
05F0 76        678          fcb     'v'     ; 2A  V
05F1 66        679          fcb     'f'     ; 2B  F
05F2 74        680          fcb     't'     ; 2C  T
05F3 72        681          fcb     'r'     ; 2D  R
05F4 35        682          fcb     '5'     ; 2E  5 or %
05F5 00        683          fcb     $00     ; 2F
               684
05F6 00        685          fcb     $00     ; 30
05F7 6E        686          fcb     'n'     ; 31  N
05F8 62        687          fcb     'b'     ; 32  B
05F9 68        688          fcb     'h'     ; 33  H
05FA 67        689          fcb     'g'     ; 34  G
05FB 79        690          fcb     'y'     ; 35  Y
05FC 36        691          fcb     '6'     ; 36  6 or ^
05FD 00        692          fcb     $00     ; 37
05FE 00        693          fcb     $00     ; 38
05FF 00        694          fcb     $00     ; 39
0600 6D        695          fcb     'm'     ; 3A  M
0601 6A        696          fcb     'j'     ; 3B  J
```

```
0602 75         697          fcb      'u'     ; 3C  U
0603 37         698          fcb      '7'     ; 3D  7 or &
0604 38         699          fcb      '8'     ; 3E  8 or *
0605 00         700          fcb      $00     ; 3F
                701
0606 00         702          fcb      $00     ; 40
0607 2C         703          fcb      ','     ; 41  , or <
0608 6B         704          fcb      'k'     ; 42  K
0609 69         705          fcb      'i'     ; 43  I
060A 6F         706          fcb      'o'     ; 44  O
060B 30         707          fcb      '0'     ; 45  0 or )
060C 39         708          fcb      '9'     ; 46  9 or (
060D 00         709          fcb      $00     ; 47
060E 00         710          fcb      $00     ; 48
060F 2E         711          fcb      '.'     ; 49  . or >
0610 2F         712          fcb      '/'     ; 4A  / or ?
0611 6C         713          fcb      'l'     ; 4B  L
0612 3B         714          fcb      ';'     ; 4C  ; or :
0613 70         715          fcb      'p'     ; 4D  P
0614 2D         716          fcb      '-'     ; 4E  - or _
0615 00         717          fcb      $00     ; 4F
                718
0616 00         719          fcb      $00     ; 50
0617 00         720          fcb      $00     ; 51
0618 27         721          fcb      $27     ; 52  ' or "
0619 00         722          fcb      $00     ; 53
061A 5B         723          fcb      '['     ; 54  [ or {
061B 3D         724          fcb      '='     ; 55  = OR +
061C 00         725          fcb      $00     ; 56
061D 00         726          fcb      $00     ; 57
061E 00         727          fcb      $00     ; 58  Caps Lock
061F 00         728          fcb      $00     ; 59  Right Shift
0620 0D         729          fcb      $0D     ; 5A  Enter
0621 5D         730          fcb      ']'     ; 5B  ] or }
0622 00         731          fcb      $00     ; 5C
0623 5C         732          fcb      '\'     ; 5D  \ or |
0624 00         733          fcb      $00     ; 5E
0625 00         734          fcb      $00     ; 5F
                735
0626 00         736          fcb      $00     ; 60
0627 00         737          fcb      $00     ; 61
0628 00         738          fcb      $00     ; 62
0629 00         739          fcb      $00     ; 63
062A 00         740          fcb      $00     ; 64
062B 00         741          fcb      $00     ; 65
062C 08         742          fcb      $08     ; 66  Backspace
062D 00         743          fcb      $00     ; 67
062E 00         744          fcb      $00     ; 68
062F 31         745          fcb      '1'     ; 69  NUM - 1 or END
0630 00         746          fcb      $00     ; 6A
0631 34         747          fcb      '4'     ; 6B  NUM - 4 or LEFT
0632 37         748          fcb      '7'     ; 6C  NUM - 7 or HOME
0633 00         749          fcb      $00     ; 6D
0634 00         750          fcb      $00     ; 6E
0635 00         751          fcb      $00     ; 6F
                752
0636 30         753          fcb      '0'     ; 70  NUM - 0 or INS
0637 2E         754          fcb      '.'     ; 71  NUM - . or DEL
```

```
0638 32    755          fcb     '2'     ; 72  NUM - 2 or DOWN
0639 35    756          fcb     '5'     ; 73  NUM - 5
063A 36    757          fcb     '6'     ; 74  NUM - 6 or RIGHT
063B 38    758          fcb     '8'     ; 75  NUM - 8 or UP
063C 1B    759          fcb     $1B     ; 76  ESC
063D 00    760          fcb     $00     ; 77  NUM LOCK
063E 00    761          fcb     $00     ; 78  F11
063F 2B    762          fcb     '+'     ; 79  NUM - + (Plus)
0640 33    763          fcb     '3'     ; 7A  NUM 3 or PAGE DOWN
0641 2D    764          fcb     '-'     ; 7B  NUM - - (Minus)
0642 2A    765          fcb     '*'     ; 7C  NUM - *
0643 39    766          fcb     '9'     ; 7D  NUM - 9 or PAGE UP
0644 00    767          fcb     $00     ; 7E  SCROLL LOCK
0645 00    768          fcb     $00     ; 7F
           769
0646 00    770          fcb     $00     ; 80
0647 00    771          fcb     $00     ; 81
0648 00    772          fcb     $00     ; 82
0649 00    773          fcb     $00     ; 83  F7
064A 00    774          fcb     $00     ; 84
064B 00    775          fcb     $00     ; 85
064C 00    776          fcb     $00     ; 86
064D 00    777          fcb     $00     ; 87
064E 00    778          fcb     $00     ; 88
064F 00    779          fcb     $00     ; 89
0650 00    780          fcb     $00     ; 8A
0651 00    781          fcb     $00     ; 8B
0652 00    782          fcb     $00     ; 8C
0653 00    783          fcb     $00     ; 8D
0654 00    784          fcb     $00     ; 8E
0655 00    785          fcb     $00     ; 8F
           786
           787  ************************************************
           788  *                                              *
           789  *   Shift - Lookup Table Used when Shift Pressed    *
           790  *                                              *
           791  ************************************************
           792
0656 00    793  shift   fcb     $00     ; 00
0657 00    794          fcb     $00     ; 01  F9
0658 00    795          fcb     $00     ; 02
0659 00    796          fcb     $00     ; 03  F5
065A 00    797          fcb     $00     ; 04  F3
065B 00    798          fcb     $00     ; 05  F1
065C 00    799          fcb     $00     ; 06  F2
065D 00    800          fcb     $00     ; 07  F12
065E 00    801          fcb     $00     ; 08
065F 00    802          fcb     $00     ; 09  F10
0660 00    803          fcb     $00     ; 0A  F8
0661 00    804          fcb     $00     ; 0B  F6
0662 00    805          fcb     $00     ; 0C  F4
0663 09    806          fcb     $09     ; 0D  TAB
0664 7E    807          fcb     '~'     ; 0E  ` or ~
0665 00    808          fcb     $00     ; 0F
           809
0666 00    810          fcb     $00     ; 10
0667 00    811          fcb     $00     ; 11  Left ALT
0668 00    812          fcb     $00     ; 12  Left SHIFT
```

```
0669 00         813          fcb      $00     ; 13
066A 00         814          fcb      $00     ; 14  Left Ctrl
066B 51         815          fcb      'Q'     ; 15  Q
066C 21         816          fcb      '!'     ; 16  1 or !
066D 00         817          fcb      $00     ; 17
066E 00         818          fcb      $00     ; 18
066F 00         819          fcb      $00     ; 19
0670 5A         820          fcb      'Z'     ; 1A  Z
0671 53         821          fcb      'S'     ; 1B  S
0672 41         822          fcb      'A'     ; 1C  A
0673 57         823          fcb      'W'     ; 1D  W
0674 40         824          fcb      '@'     ; 1E  2 or @
0675 00         825          fcb      $00     ; 1F
                826
0676 00         827          fcb      $00     ; 20
0677 43         828          fcb      'C'     ; 21  C
0678 58         829          fcb      'X'     ; 22  X
0679 44         830          fcb      'D'     ; 23  D
067A 45         831          fcb      'E'     ; 24  E
067B 24         832          fcb      '$'     ; 25  4 or $
067C 23         833          fcb      '#'     ; 26  3 or #
067D 00         834          fcb      $00     ; 27
067E 00         835          fcb      $00     ; 28
067F 20         836          fcb      ' '     ; 29  Space
0680 56         837          fcb      'V'     ; 2A  V
0681 46         838          fcb      'F'     ; 2B  F
0682 54         839          fcb      'T'     ; 2C  T
0683 52         840          fcb      'R'     ; 2D  R
0684 25         841          fcb      '%'     ; 2E  5 or %
0685 00         842          fcb      $00     ; 2F
                843
0686 00         844          fcb      $00     ; 30
0687 4E         845          fcb      'N'     ; 31  N
0688 42         846          fcb      'B'     ; 32  B
0689 48         847          fcb      'H'     ; 33  H
068A 47         848          fcb      'G'     ; 34  G
068B 59         849          fcb      'Y'     ; 35  Y
068C 5E         850          fcb      '^'     ; 36  6 or ^
068D 00         851          fcb      $00     ; 37
068E 00         852          fcb      $00     ; 38
068F 00         853          fcb      $00     ; 39
0690 4D         854          fcb      'M'     ; 3A  M
0691 4A         855          fcb      'J'     ; 3B  J
0692 55         856          fcb      'U'     ; 3C  U
0693 26         857          fcb      '&'     ; 3D  7 or &
0694 2A         858          fcb      '*'     ; 3E  8 or *
0695 00         859          fcb      $00     ; 3F
                860
0696 00         861          fcb      $00     ; 40
0697 3C         862          fcb      '<'     ; 41  , or <
0698 4B         863          fcb      'K'     ; 42  K
0699 49         864          fcb      'I'     ; 43  I
069A 4F         865          fcb      'O'     ; 44  O
069B 29         866          fcb      ')'     ; 45  0 or )
069C 28         867          fcb      '('     ; 46  9 or (
069D 00         868          fcb      $00     ; 47
069E 00         869          fcb      $00     ; 48
069F 3E         870          fcb      '>'     ; 49  > or .
```

```
06A0 3F        871         fcb     '?'     ; 4A  / or ?
06A1 4C        872         fcb     'L'     ; 4B  L
06A2 3A        873         fcb     ':'     ; 4C  ; or :
06A3 50        874         fcb     'P'     ; 4D  P
06A4 5F        875         fcb     '_'     ; 4E  - or _
06A5 00        876         fcb     $00     ; 4F
               877
06A6 00        878         fcb     $00     ; 50
06A7 00        879         fcb     $00     ; 51
06A8 22        880         fcb     $22     ; 52  ' or "
06A9 00        881         fcb     $00     ; 53
06AA 7B        882         fcb     '{'     ; 54  [ or {
06AB 2B        883         fcb     '+'     ; 55  = OR +
06AC 00        884         fcb     $00     ; 56
06AD 00        885         fcb     $00     ; 57
06AE 00        886         fcb     $00     ; 58  Caps Lock
06AF 00        887         fcb     $00     ; 59  Right Shift
06B0 0D        888         fcb     $0D     ; 5A  Enter
06B1 7D        889         fcb     '}'     ; 5B  ] or }
06B2 00        890         fcb     $00     ; 5C
06B3 7C        891         fcb     '|'     ; 5D  \ or |
06B4 00        892         fcb     $00     ; 5E
06B5 00        893         fcb     $00     ; 5F
               894
06B6 00        895         fcb     $00     ; 60
06B7 00        896         fcb     $00     ; 61
06B8 00        897         fcb     $00     ; 62
06B9 00        898         fcb     $00     ; 63
06BA 00        899         fcb     $00     ; 64
06BB 00        900         fcb     $00     ; 65
06BC 08        901         fcb     $08     ; 66  Backspace
06BD 00        902         fcb     $00     ; 67
06BE 00        903         fcb     $00     ; 68
06BF 31        904         fcb     '1'     ; 69  NUM - 1 or END
06C0 00        905         fcb     $00     ; 6A
06C1 34        906         fcb     '4'     ; 6B  NUM - 4 or LEFT
06C2 37        907         fcb     '7'     ; 6C  NUM - 7 or HOME
06C3 00        908         fcb     $00     ; 6D
06C4 00        909         fcb     $00     ; 6E
06C5 00        910         fcb     $00     ; 6F
               911
06C6 30        912         fcb     '0'     ; 70  NUM - 0 or INS
06C7 2E        913         fcb     '.'     ; 71  NUM - . or DEL
06C8 32        914         fcb     '2'     ; 72  NUM - 2 or DOWN
06C9 35        915         fcb     '5'     ; 73  NUM - 5
06CA 36        916         fcb     '6'     ; 74  NUM - 6 or RIGHT
06CB 38        917         fcb     '8'     ; 75  NUM - 8 or UP
06CC 1B        918         fcb     $1B     ; 76  ESC
06CD 00        919         fcb     $00     ; 77  NUM LOCK
06CE 00        920         fcb     $00     ; 78  F11
06CF 2B        921         fcb     '+'     ; 79  NUM - + (Plus)
06D0 33        922         fcb     '3'     ; 7A  NUM 3 or PAGE DOWN
06D1 2D        923         fcb     '-'     ; 7B  NUM - - (Minus)
06D2 2A        924         fcb     '*'     ; 7C  NUM - *
06D3 39        925         fcb     '9'     ; 7D  NUM - 9 or PAGE UP
06D4 00        926         fcb     $00     ; 7E  SCROLL LOCK
06D5 00        927         fcb     $00     ; 7F
               928
```

```
 06D6 00         929          fcb      $00     ; 80
 06D7 00         930          fcb      $00     ; 81
 06D8 00         931          fcb      $00     ; 82
 06D9 00         932          fcb      $00     ; 83  F7
 06DA 00         933          fcb      $00     ; 84
 06DB 00         934          fcb      $00     ; 85
 06DC 00         935          fcb      $00     ; 86
 06DD 00         936          fcb      $00     ; 87
 06DE 00         937          fcb      $00     ; 88
 06DF 00         938          fcb      $00     ; 89
 06E0 00         939          fcb      $00     ; 8A
 06E1 00         940          fcb      $00     ; 8B
 06E2 00         941          fcb      $00     ; 8C
 06E3 00         942          fcb      $00     ; 8D
 06E4 00         943          fcb      $00     ; 8E
 06E5 00         944          fcb      $00     ; 8F
                 945
 06E6            946          end
                 947
                 948
                 949
                 950

 Symbol Table

ALTDEC           053D
ALT_REL          0554
ASC              00C3
BYTE             00C0
CANCEL           0386
CAPLOC           0002
CAPLOCK          0002
CAPS             03B5
CAPS_ON          038C
CLKIN            0001
CLKOUT           0003
CLRASC           0588
CLR_PAR          04C7
COMPLET          0585
CTRLHEX          0523
CTRL_RE          0572
DATA             059E
DATAIN           0000
DATAOUT          0002
DELAY            05C2
ERROR            04DA
EXTEND           03D6
EXTEND1          03E2
EXTEND2          03EB
EXTEND3          03F4
EXTEND4          0410
EXTEND5          041B
EXTEND6          0426
HE1_RAN          0530
HIGHLOW          051C
LALT             0004
LCTRL            0006
LED              00C2
```

```
LEDSHOW         047B
LOOP            04AC
LSHIFT          0000
MAIN            0322
MAIN1           032E
MAIN10          0375
MAIN11          037C
MAIN12          0395
MAIN13          03A2
MAIN14          03AC
MAIN2           0335
MAIN3           033C
MAIN4           0342
MAIN5           0348
MAIN6           0351
MAIN7           035A
MAIN8           0363
MAIN9           036C
MARK            04B5
NEXT            04B8
NOSHIFT         05C6
NRESET          0004
NUMLOCK         0001
NUMS            03C0
OUTRANG         0551
PNUMLCK         0006
PSCRLCK         0007
RALT            0005
RCTRL           0007
RECDATA         04ED
RECEIVE         04E2
RECNEXT         04FD
RECSET          04FA
RELEA10         045E
RELEAS3         042E
RELEAS4         0434
RELEAS5         043A
RELEAS6         0443
RELEAS7         044C
RELEAS8         0452
RELEAS9         0458
RELEASE         0429
REL_EX2         046F
REL_EX3         0478
REL_EXT         0461
RESET           0317
RETURN          03B2
RS232T          0591
RSHIFT          0001
RSMARK          05A7
RSNEXT          05AB
RSTFLAG         031E
R_ERROR         0512
SCRL            03CB
SCRLOCK         0000
SET_PAR         04C2
SHIFT           0656
SHIFTON         0392
```

```
SPACE           04B0
START           0300
STATUS          00C1
STORE           0547
TRANSMIT        0497
TR_ACKN         04C9
TXD             0005
```