PC Keyboard FAQ

Contents:

[Document Version: **1.01a**] [Last Updated: **6/29/97**]

Chapter 1) Mark Schultz's AT keyboard interface

1.1) About the Author

Author: Mark Schultz
E-Mail: mschultz@wctc.net
Version: 1.00
Date: 16 Aug 1995

Corrections from Stuart Ferguson (www.microwizard.com)

1.2) Disclaimer & Feedback Request

This writeup is by far from complete. As I mentioned earlier, I originally started this as a 'simple' (!) response to a query posted here. Little did I know that I'd be here typing away for better than an hour. Given the trouble I had in acquiring info as to how to control and utilize PC keyboards in non-PC applications (which is still incomplete), I can appreciate the frustration experienced by others in obtaining this info. If the demand justifies it, I would be willing to create a more formal, detalied description of the PC/AT keyboard protocol along with code samples in 68HC05 assembly language. If you are interested in such a writeup, respond to this message indicating your wishes or (preferably) respond via E-mail to mschultz@wctc.net

---

1.3) Introduction

I might be able to help you out with a few general pointers, as I have built a few homebrew projects that have employed AT-style PC keyboards. I appreciate your plight; getting the AT keyboard to work in my first project that employed it was a gut-wrenching, hair-pulling experience.

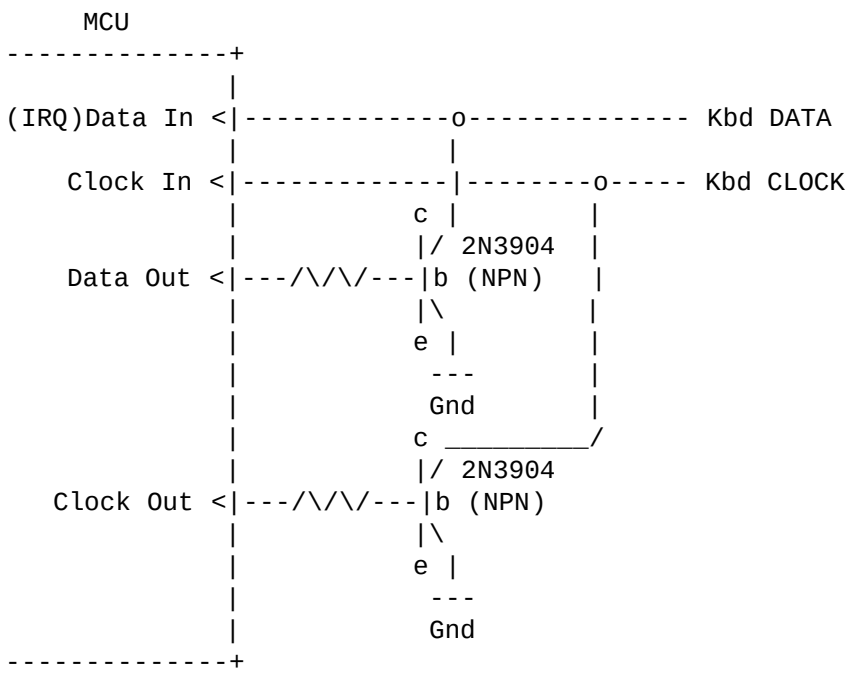
1.4) Description

The AT keyboard transmission protocol is clocked serial (with the keyboard as clock source or 'master'), 11 bits in length. One start bit (logic 0), 8 data bits (LSB first), one odd parity bit and a stop bit (logic 1). The clock rate is approx. 10-20 KHz and can vary from keyboard to keyboard. The keyboard data format resembles 8-odd-1 asynchronous transmission format. However, the bit rate from keyboard to keyboard can vary significantly so it is necessary to use a clocked serial interface (SPI in Motorola parlance) or a async (SCI) interface with a 1x receive clock input. For MPUs that cannot accomodate this transmission protocol directly using a hardware subsystem (SPI or SCI) I have found it useful to tie the CLOCK input to a IRQ (external interrupt) line and read the data bits one-by-one on the falling edge of each clock pulse.

Both CLOCK and DATA lines are implemented on the keyboard end as open-collector outputs with pull-up resistors to +5V. It is possible (and sometimes necessary) for the host system to actively pull these lines LOW. In the systems I have built, I have used 4 MCU I/O lines to control the keyboard: Two inputs

(for clock and data in) as well as two outputs connected via limiting resistors to NPN transistors to allow the MCU to drive the CLOCK and DATA lines of the keyboard. The following crude diagram attempts to illustrate the basic circuit:

1.5) Schematic



1.6) Keyboard to system protocol

The keyboard will transmit keystroke data to the host system as soon as a key is pressed (or released) if both the DATA and CLOCK lines are HIGH. If the CLOCK line is LOW, the keyboard will buffer keystroke data until the CLOCK line goes HIGH again (the clock line acts as a -RTS signal). If the DATA line is LOW, the keyboard prepares to accept a 11-bit control message from the host (see below).

If the CLOCK line is pulled LOW by the host while the keyboard is transmitting data (up to the 10th bit) for at least 60 uS, the keyboard will suspend transmission and prepare to receive a system message. The keyboard will eventually re-transmit the data byte that was interrupted, unless the keyboard was successful in transmitting the 10th bit, in which case the keyboard considers the data byte as successfully sent. It is generally a bad idea to interrupt keyboard transfers this way.

1.7) System to keyboard protocol

AT-style keyboards are capable of accepting control messages from the system in addition to sending scan code information to the host. In this way, the status LEDs may be controlled and the keyboard typematic parameters (repeat delay and rate) can be set. Command transmission to the keyboard is initiated by bringing the keyboard CLOCK line LOW for at least 60 uS. After the 60 uS delay, the DATA line should be brought LOW and the CLOCK line released (HIGH). Make sure to bring the DATA line LOW before releasing the CLOCK line. Some time later (up to 10 milliseconds) the keyboard will start to generate clock signals. At each HIGH to LOW clock transition the keyboard will clock in a new bit. The data stream from the system (on the DATA line) should conform to the serial protocol described above.

After the parity bit has been clocked out, the host should release (bring HIGH) the DATA line and wait for the keyboard to send another CLOCK pulse (this will be the 10th HIGH to LOW transition, not counting the initial H->L transition generated by the host). The KEYBOARD will then bring the DATA line LOW sometime before the 11th HIGH to LOW transition of the CLOCK line to acknowledge reception of the command byte.

If the DATA line is not released (allowed to go HIGH) after the 10th CLOCK bit then the keyboard will continue to issue clock pulses until the DATA line is released. In this event, the keyboard will (after some delay) pull the DATA line LOW and transmit a RESEND status byte (FEh).

1.8) Commands - Host to keyboard

(all numeric values are in HEX)

I will present these command bytes briefly here. I will be happy to expand on their function and purpose if asked. This started out as a simple reply to a question and has almost turned into a FAQ <g>. Depending on demand, I just might expand this impromptu reply into a full-fledged FAQ.

F5
: Default Disable. Resets keyboard, returns ACK and suspends scanning, waiting for another command. Does not affect the indicator LEDs.

EE
: Echo. Responds with ECHO code (EE).

F4
: Enable. Clears output buffer, enabled kbd, returns ACK.

F2
: Read ID. Responds with ACK and two ID bytes (83,AB). Resumes scanning even if previously disabled.

FE
: Resend. Retransmit last sent scan code.

FF
: Reset. Resets keyboard CPU, starts power-on test. Responds with power-on-test byte.

F0
: Select scan code set. Responds with ACK, then waits for host to send a byte (01,02 or 03) specifying the scan code set to use. If 00 is sent, keyboard responds with ACK followed by the scan code set in use.

F7
: 'Set all keys typematic'

F8
: 'Set all keys make/break'

F9
: 'Set all keys make'

FA
: 'Set all keys typematic/make/break' I am not certain what these commands do. I suspect they control the way the keyboard transmits scan codes. All of the above commands respond with an ACK code.

F6
: Set default. Responds as the 'Default Disable' command, but does not inhibit scanning. Does not affect LED indicators.

FB
: 'Set key type typematic'

FC

'Set key type make/break'

FD
: 'Set key type make' Again, I'm not sure what these do. Unlike the 'set all' commands, these commands presumably act on single keys. The keyboard sends the ACK code, then waits for keyboard scan code(s). ACK is set after each scan code is received. Keyboard remains in the 'set type' state until a new command is received.

ED
: Set/reset status indicators. This command allows you to control the status LEDs on the keyboard. Keyboard responds with ACK and waits for a option byte, bitmapped as follows: b0-Scrollock, b1-Numlock, b2-Capslock, b3..7=0. A '1' bit turns the indicator ON.

F3
: Set autorepeat rate/delay. Responds with ACK, then waits for an option byte that specifies the autorepeat delay and rate, bitmapped as follows: b7-unused. b6..5-Repeat delay (00=250 mS, 11=1000mS). b4..0-Repeat rate (00000=30x/sec, 11111=2x/sec). Keyboard responds with ACK after reception of the option byte.

1.9) Keyboard response codes

FA
: ACK (acknowledge), response to most commands.

AA
: Poweron test successful.

EE
: Echo. Response to the ECHO command (EE).

00 or FF
: Key detection error, also indicates buffer overflow.

FE
: Resend. Sent in response to the receipt of a invalid command code or code with a bad parity bit.

The keyboard can, of course, send key scan make/break codes as well as numeric responses to certain commands as outlined above.

1.10) References

Portions of the information contined here were taken from _101/102RX Series Electrical Specifications, Honeywell Keyboard Specifications_ published by Honeywell Keyboard Division, 4171 N. Mesa, Bldg. D, El Paso, TX 79902.

Chapter 2) Assorted info on XT/AT/PS2 Keyboard Interface

2.1) Introduction

(From the Editor)

Here's some assorted information about the PC keyboard. Look through the index and see if you see anything interesting. Be aware that sometimes the Index entry doesn't entirely describe the article! That's life. Sorry.

If you have anything that doesn't quite fit into an existing keyboard FAQ, please send it to me and I'll include it here.

2.2) Booting PC without keyboard

(From Tony Snyder)

A few days ago someone needed information about booting a PC without a keyboard. I didn't see much posted, that directly satisfied his needs, other than some advice regarding reworking his bios etc. I just ran across an ad in the September, 1993 Circuit Cellar INK, The Computer Applications Journal, issue number 38, that could help. On page 93, Verta Systems Corporation, 27 Newtown Rd, Plainview, NY, tel # (516)454-6469, advertises a keyboard eliminator which plugs into the keyboard port. Its called the Phantom Keyboard. That's all I have for now.

2.3) IBM Keyboard Interfact Project

(From Eric Rudolph)

IBM Keyboard Interfact Project, by Eric Rudolph, 1991.
No Copyright Whatsoever, but I would like credit where it's due.

A little background on the IBM AT keyboard:

The keyboard I used for this circuit was a BTC 5339sx. It costs about $40 Lucky Computers, 1-800-348-5825. I have also tested the circuit on a standard PC keyboard, and an HP Vectra AT keyboard.

The AT keyboard DIN connector has 5 pins. Pin 3 is called Reset, but it's reserved, so we can't use it. Forget it exists on the AT keyboard. They just charge you money for it. (joke) Open collectors drive the clock and data pins, so when they are not driven low, they float at 5 volts.

The keyboard transmits data by bits, each synchonous somehow with the clock line. They keyboard when clocking in or out data, always runs the clock. The controller can drive the clock line, but not with reference to data.

When the Keyboard sends data, it first sets the data, then drives clock low, then high, and then changes the data to the next value.

The AT uses 11 bits for a transmission, 1 start bit (0), 8 data bits, 1 parity bit-set if the number of 1's in the data bits is even, and a stop bit (1). When the keyboard sends it's last bit, the stop bit, the controller must drive clock line low as a handshake and to tell the keyboard not to send until clock goes high again.

Theoretically, the controller could interrupt the sending of the bits, but I consider this unnecessary, and don't bother with it. When the computer needs to send a command to the keyboard, it sets clock line high and the data line low. When the keyboard sees this, it will start clocking pulsed on the data line.

Then, the controller must look for the clock line going low, set the data bit, wait for the clock to go high, then wait for the clock to go low again, and then change the bit. Thus, it changes the data in the middle of

the clock low pulse. When the keyboard has received it's 10th bit, it will drive the data line low while at the same time clocking out an extra clock low pulse. Then, it expects a handshake of the clock line low from the controller.

```
0       1       2   |   7            stop    extra
Clock:------------\___/---\___/---\___/---\___/---\___/---\___/---\___/end

Data:-------\_______00000001111111122222|6677777777PPPPPPPP1111xxxxxxxx_____
```

Notice there is NO start bit when the controller sends data to the Keyboard.

2.3.1) Special Commands the Keyboard can Send to the Controller

```
00      Keyboard buffer overflowed
AA      Selftest passed
FA      The command sent was received correctly
FE      The command sent was received poorly. Please resend.
```

2.3.2) Special Commands the Controller can Send to the Keyboard

```
ED      Set the LEDs according to next byte I send
        bit 0=Scroll lock 1=on
        bit 1=Num lock
        bit 2=Caps lock
        bits 3-7 must be 0
F4      clear the key buffer and start scanning
F6      restore default values
FE      retransmit last character, please
FF      Reset, you stupid keyboard!
```

Whew! That about raps it up for the AT keybard! Enough said, right?

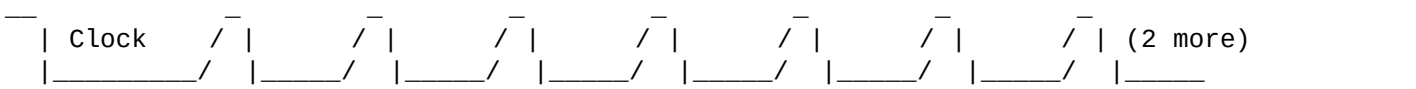
XT keyboards transmit much the same way except they only use 10 bits. Two start bits (both high) and 8 data bits, transmitted in order 0-1-2...-7 The last bit is a make/break bit which is 1 to signify a break.

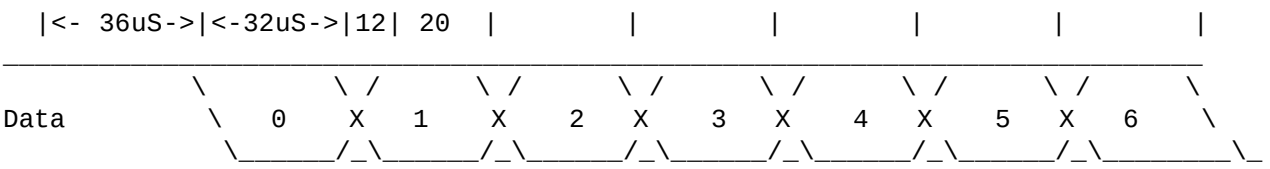
The XT can have no commands sent to it. The way to reset it is to drive the Clock line low for some longish period of time. The keyboard will not send data (it will hold off) if the data line is being held low by an external source (the controller)

2.4) IBM keyboard port specs wanted

(From Tomi H Engdahl)

The only data I have left from hacking at my keyboard when it died a couple of years ago are some timing diagrams recorded from a 'scope (wretched ASCII graphics follow):





This is for an XT keyboard.

The clock and data lines are both normally high. When a key is pressed, Clock goes low for 36uS. This is followed by 9 clock pulses, each 12.5uS low and 20uS high; on the 10th rising edge the clock stays high until the next sequence. The Data line changes on the rising edge of the clock. There are only 7 data bits (the last two bit times on the Data line are always low). The ~32uS clock period gives a ~31KHz bit rate. At least, this is how I interpret my drawings...

2.5) AT Keyboard Interface INFO

(From Bill Mayhew)

The IBM-PC-compatible keyboard interface is fairly nasty to duplicate. I've thought several times about using PC keyboards as input devices, but have just fallen back to ASCII parallel keyboards or making my own scanning matricx with something like a KR2376.

The problem with the IBM-PC keyboard is that it uses a bidirectional protocol with both the PC's on-board 8741 (or equivalent) keyboard controller single-chip microprocessor and the keyboard toggling the clock line and exchanging pulses and scan codes with relatively strict timing requirements.

I figured my time to design my own version of the handshake would be more expensive than obtaining a more easily dealt-with keyobard. If you are talking volume use or have to use the keyboard input for some reason, than you can nix the timing argument.

What would be real handy would be for somebody to produce an 8741 that is programmed to deal with the keyboard and output the eqiivalent ASCII decoded from the scan codes along with a strobe pulse. It would make a nice handy experimenter's aid. Maybe somebody already has one?...

If you want the gory timing details, obtain an IBM PC-AT technical refrence manual. It should have a lot of details on the keyboard interface. I've got the Model-80 technical ref and it has a lot of info on the keyboard interface. IBM has many technical manuals for fairly reasonable prices. My old book listed thier number as 800-IBM-PCTB. Taht is several years old, so the number may well have changed by now.

(From Lance Bresee)

I probably should have Replied to the original poster, but I thought others might be interested as pc keyboards are cheap.

MicroCornucopia
No 52
March-April 1990
Page 36 - 43
An AT Keyboard Interface
Don Rowe

Includes description of AT Keyboard serial data transfer and a circuit to interface to an AT keyboard.

Here is the wiring for both:

2.6.1) 5 Pin DIN (AT/XT)

```
1   CLK
2   DATA
3   NOT RESET
4   GND
5   +5V
```

6 Pin PS2

(From Tomi Engdahl)

I have always been in impression that PS/2 keyboard pin numbering goes like this (but I might be also wrong).

```
        4 u 6
       1  .  2
        3   5

1.      GND
2.      +5V
3.      DATA
4.      CLOCK
5.      Not used
6.      Not used
```

The PS2 is numbered as follows:

```
        ^
      6   5
     4     3
     > 2 1 <

1   DATA
2   No connection
3   GND
4   +5V
5   CLK
6   No connection
```

The <>'s indicate the position of the triangular cutouts. The 5 Pin DIN is standard universal layout.

Also, see the "Pinouts for various connectors in Real Life(tm)" (Sub-ToC) for other pinouts.

2.7) XT vs. AT keyboard

(From Bruce Adler)

The XT and AT keyboards are NOT compatible. The only way a XT keyboard will work on an AT computer is if you've got a motherboard and BIOS which are specifically designed to support both keyboard types.

The keyboards differ in at least the following ways:

1. The XT kb. generates 2 start bits, 8 data bits, make/break bit, and a stop bit. The AT kb. is 1 start bit, 8 data, 1 parity, and a stop bit.
2. The XT uses a make/break bit to indicate key up/down. The AT sends one byte for keydown and two bytes for keyup.
3. The XT keyboard scan codes have different values than then AT keyboard make/break codes (for corresponding key locations).
4. The XT keyboard doesn't accept any of the command strings accepted by the AT keyboard.
5. The XT keyboard is reset by fiddling the clock and data lines; the AT keyboard accepts a reset command string.

In the same manner plugging an AT keyboard into an XT computer won't work either, unless you have one of those clone keyboards which allow you to select XT compatible mode.

2.8) BIOS code modifications

(From Byron A Jeff)

On any SimTel archive you can find BIOS for the keyboard controller. Look in directory:

...msdos/sysutl

For file:

bios-asm.zip "Public domain generic PC BIOS (MASM source)"

2.9) IBM-PC keyboard interfacing

(From Brad Siim)

[...] The PS2 keyboard interface is not a very friendly or easy to understand interface. The best place to get information is in IBM's technical reference manual. All of the documentation that I have seen on this interface is the same/similar (ie. Compac's Technical Reference spec) and almost seems to be a direct copy of IBM's reference manual.

The core of ALL/most PS2 keyboard interfaces is the Intel 8042 microcontroller or some derivative of it. You will need to get a copy of this datasheet as well as IBM's keyboard spec.

Basic Description:

1. The 8042 side of the interface is two bidirectional open collector lines called CLOCK and DATA.
2. The KEYBOARD side of the interface is identical.
3. Both sides can drive the lines and often do so at the same time. Each side has to figure out what is going on if the other side has decided to interrupt.
4. When the system CPU wants to send something to a KEYBOARD device it writes a command to the 8042 which tells the 8042 that the next command written to the 8042 should be passed on to the KEYBOARD device across the serial DATA line.
5. Assume that when idle both the DATA and CLOCK lines are high.
6. The 8042 pulls the clock line low to inhibit any transmission from the KEYBOARD.

7. The 8042 pulls the data line low to get the KEYBOARD's attention to the fact that it wants to transmit.
8. The 8042 release the CLOCK line and waits for the KEYBOARD to pull the clock line low. When the CLOCK line has been pulled low the 8042 places its first bit of data on the DATA line.
9. The keyboard toggles the clock line and clocks data across on the data line. The 8042 controller will place a new bit on the data line each time the clock is pulled LOW by the KEYBOARD.

   The following picture will explain better:

```
CK HHllllHHLLLHHHLLLHHHLLLHHHLLLHHHLLLHHHLLLHHHLLLHHHLLLHHHLLLHHHLLLHHHLLLHllll
DA HHHlllllll0000001111112222223333334444445555556666667777777ppppppssssssLLLLHHH
        ^                                                                    ^
    start bit                                                          important bit

where:  l = 8042 driving the line
        L = KEYBOARD driving the line
        1..7 data
         = Parity 8042 driving
        s = stop   8042 driving
         = PARITY KEYBOARD driving the line
        S = STOP   KEYBOARD driving the line
```

10. The last bit in the last clock cycle is a data bit that is returned by the KEYBOARD. The keyboard may also be required to return a whole byte/bytes to the controller depending upon the data that was sent by the controller to the KEYBOARD. After the controller has seen the last bit it usually will inhibit the keyboard by yanking the clock line low. Please note that if the keyboard has to return data to the 8042 it will do so under the following protocal after the 8042 has released the clock line.
11. During the above transmission the contoller may abort the operation by holding the CLOCK line low. The keyboard has to montitor the CLOCK line and stop clocking if it sees that the CLOCK line is not going high.
12. If the KEYBOARD wants to send data: Assume both lines are high. If they are not high the keyboard can't send. If the clock line is being held low by the controller, the controller does not want data. If the data line is being held low by the controller the controller wants to send data and the keyboard must start clocking it.
13. So the lines are high: The keyboard pulls the data line low and then pulls the clock line low and then back to high. The data line is changed while the clock is HIGH. The KEYBOARD pulls the clock line low and high and repeats the operation as follows:

```
CK HHHLLLHHHLLLHHHLLLHHHLLLHHHLLLHHHLLLHHHLLLHHHLLLHHHLLLHHHLLLHHHLLLHHHLLllllll
DA HLLLLLL0000001111112222223333334444445555556666667777777PPPPPPssssssLLLLLHHHH
      ^
    start bit
```

I have explained this in detail because the documentation is VERY hard to decripher. I hope this is accurate and helps you. I have not discussed timing, implementation or voltage levels as I'm sure you will not have trouble with these aspects.

Chapter 3) Richard Steven Walz's IBM PC Keyboard ScanCode FAQ

3.1) About the Author

IBM Keyboard/Scancode FAQ
Author: Richard Steven Walz
Email: rstevew@armory.com
Ver: 0.9a
Rel: 10 June 1994
Note: **Please retain this banner**

3.2) Introduction

The IBM keyboard was designed to be completely reconfigurable short of giving you keycaps with replacable transparencies under a clear top!!! Any key can be interpreted by a program as anything. They needed more than the ASCII codes for this (what's the ASCII code for 'left shift" and 'alt'?!), so they assigned scancodes whose values were arbitrary, except perhaps from a PC board layout and trace routing or keyboard controller firmware perspective. Thus these scancodes are sent by the keyboard, in time to its own clock, both of which go to the 8255 PIA port chip on the motherboard, after a bit of serial to parallel work by a couple other chips.

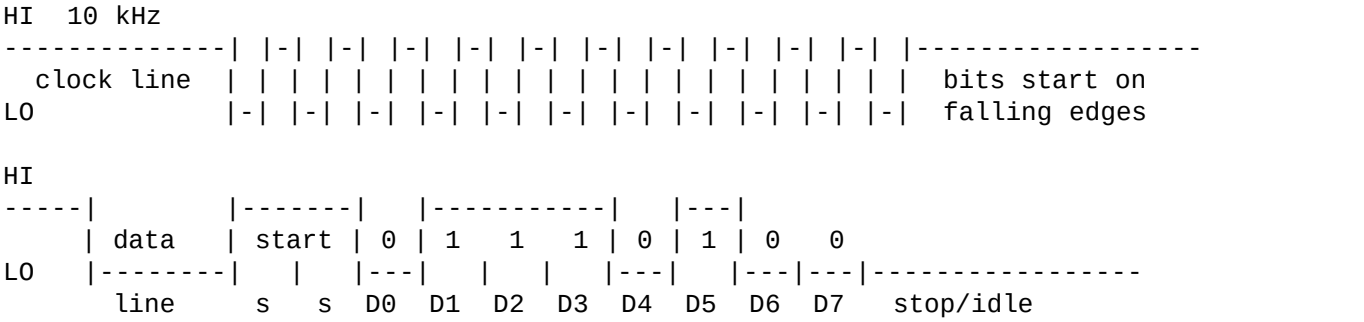
The keyboard clock and data lines are open collector, meaning that they are pulled up with resistors and can be pulled down by either end of the line to for communication in either direction. Thus there is a protocol that each end observes about their logic states.

The keyboard monitors the state of the data and clock lines prior to any send to the computer. If the clock line is LO, (TTL LO or near 0 Volts), then the keyboard is disabled and will not send. If the clock line is high, (TTL HI or getting close to but not greater than 5 Volts), and the data line is LO, then input TO the keyboard is accepted FROM the computer!

Finally it should be mentioned that the keyboard only starts a send to the computer when the data line and the clock line are high. This means that the computer is not trying to talk to the keyboard or trying to make the keyboard wait to send. When the two lines are high, since the lines are open collector and pulled up passively, the keyboard can take them over and start the clock and pull the data line low. On the falling edge of the 10 kHz clock, the keyboard again raises the data line to form two start bits of .2 ms duration, signaling the start of the byte send. Then the bits are sent in LSB first (D0) and then on up through the eight bit byte to the MSB (D7), and then the line returns to the LO state!

This is an unusual setup compared to most types of serial communications, as usually the line remains HI when not being pulled LO to make a start bit or bits, and THEN the data is sent using its correct logical value as a HI '1' or a LO '0'. Here the data is sent with its logical sense preserved, but the start bits are suddenly HI from a inactive but ready LO data line, and the bits are read by timing off the received clock on the clock line, which is also taken over by the keyboard and pulled LO to signify time to start a bit, a start bit OR a data bit.

The diagram is thus:



As was noted above, there is a state of the two lines where the computer can send data to the keyboard, and my documentation on this was more vague. If anyone can add to it, I would appreciate it. My BTC-5060 "el cheapo" terminal/AT-type keyboard manual shows a timing diagram which I believe to be fallacious, as they even have the bits in reverse order. I will want to test this keyboard with the scope to see the real story. The manual DOES, however, indicate that this keyboard will correctly operate either an XT or an AT computer and up, and it truly does! I would like to know how to make it shift to the other mode and how to control its LED states as well. This paragraph is a call for more info from out there, and more research on my part, as I see conflicting information. This should be considered a preliminary release.

This keyboard sends data not just of occurence of a key press and release but it scans the keyboard keys thousands of times a second and sends data to the computer as to whether a key has been pressed and whether it has been released, and in the order these occurred, and it stores this change data in a buffer waiting to send it as soon as the computer will accept it. This makes this keyboard extremely configurable, as we always know then what has occurred and the key presses and releases can be noted with great complexity and interpreted to our heart's content. All keys are on an equal footing and are not simply super-shift keys in the case of control, alt, and shifts. These can be read as well and used in a program. In addition, this keyboard has the IBM "typematic" action, whereby if a key is held down for more than the assigned delay, the keyboard resends its 'make' code over and over till it is released or another key takes its place as the repeater.

The scancodes for the XT are each different bytes with the 'make' byte given one hexadecimal number and the 'break' byte given the same number with its high bit (MSB) set to "1", thus the break codes are all derivable by adding 80 hexadecimal to the make code for each key. The codes have nothing to do with the ASCII value of a character seen on the keycap. They are simply codes. All values below are hexadecimal, which means to get the decimal, you can multiply the left number by 16, where the left number is a digit such that 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, and F really mean numbers 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, and 15 in decimal. Then you take the right digit and convert it straight away and add it to the multiplication of the left digits's value which was multiplied by 16. Thus the decimal equivalent for the hexadecimal for C9 is 'C' or 12 times 16, which is 192, plus '9', which is just 9, to make 201. Likewise 3F converts to '3' or 3 times 16, which is 48, plus 'F' which is 15, which add to 63. And once more for good measure, BD converts to decimal thus: 'B' is 11, and 11 times 16 is 176, and 'D' is 13, so added they make 189. The scancodes found at port 60H or 96 are the 'make' codes for those keys. In BASIC language, the first INKEY$ byte is the ASCII interpretation in decimal, unless there are two bytes for the interpretation of extended characters, then it is the second. These INKEY$ codes, while often called scancodes mistakenly, are merely extended keycodes particular to BASIC language.

3.3) The XT Scancodes

These are all hexadecimal: 'make' code top / 'break' below

F1	F2	`	1	2	3	4	5	6	7	8	9	0	-	=	\	BS	ESC	NUML	SCRL	SYSR
3B	3C	29	02	03	04	05	06	07	08	09	0A	0B	0C	0D	2B	0E	01	45	46	**
BB	BC	A9	82	83	84	85	86	87	88	89	8A	8B	8C	8D	AB	8E	81	C5	C6	

F3	F4	TAB	Q	W	E	R	T	Y	U	I	O	[	]	Home	Up	PgUp	PrtSc
3D	3E	0F	10	11	12	13	14	15	16	17	18	19	1A	1B	47	48	49	37
BD	BE	8F	90	91	92	93	94	95	96	97	98	99	9A	9B	C7	C8	C9	B7

F5	F6	CNTL	A	S	D	F	G	H	J	K	L	;	'	ENTER	Left	5	Right	-
3F	40	1D	1E	1F	20	21	22	23	24	25	26	27	28	1C	4B	4C	4D	4A
BF	C0	9D	9E	9F	A0	A1	A2	A3	A4	A5	A6	A7	A8	9C	CB	CC	CD	CA

F7	F8	LSHFT	Z	X	C	V	B	N	M	,	.	/	RSHFT	End	Dn	PgDn	+
41	42	2A	2C	2D	2E	2F	30	31	32	33	34	35	36	4F	50	51	4E
C1	C2	AA	AC	AD	AE	AF	B0	B1	B2	B3	B4	B5	B6	CF	D0	D1	CE

F9	F10	ALT	SPC	CAPLOCK	Ins	Del
43	44	38	39	3A	52	53
C3	C4	B8	B9	BA	D2	D3

Now I will explain the AT scan codes. They are done slightly differently. The AT keyboard response is a different set of events. When the key is pressed, the scan code is sent, and when the key is released, two bytes are sent, the keyboard sends F0 hex and then the scancode again, thus we will only need to list the scancodes below for the AT keyboard activity.

3.4) The AT Scancodes

Again, these are hexadecimal:

F1	F2	`	1	2	3	4	5	6	7	8	9	0	-	=	\	BS	ESC	NUML	SCRL	SYSR
05	06	0E	16	1E	26	25	2E	36	3D	3E	46	45	4E	55	5D	66	76	77	7E	84

F3	F4	TAB	Q	W	E	R	T	Y	U	I	O	[	]	Home	Up	PgUp	PrtSc
04	0C	0D	15	1D	24	2D	2C	35	3C	43	44	4D	54	5B	6C	75	7D	7C

F5	F6	CNTL	A	S	D	F	G	H	J	K	L	;	'	ENTER	Left	5	Right	-
03	0B	14	1C	1B	23	2B	34	33	3B	42	4B	4C	52	5A	6B	73	74	7B

F7	F8	LSHFT	Z	X	C	V	B	N	M	,	.	/	RSHFT	End	Dn	PgDn	+
83	0A	12	1A	22	21	2A	32	31	3A	41	49	4A	59	69	72	7A	79

F9	F10	ALT	SPC	CAPLOCK	Ins	Del
01	09	11	29	58	70	71

And that's all I know about it. If you would like to add information to this small tutorial FAQ, please send it to rstevew@armory.com for review and please don't add to this FAQ and republish it without my consent. I will keep this up to date if you will be kind and handle it that way. Thanks.

3.5) Plea

I have been looking for definitive doc on the keyboards, but everybody seems as wrong or contradicted or unsure as do I. I haven't had time to actually diagnose the behavior of keyboards with a scope or logic analyzer, so if anybody has anything that makes sense of this and the the bode.ee.ualberta.ca stuff, which I **HAVE** looked at, to little avail, please let me know!

Chapter 4) Anthony Berkow's Keyboard Basics

4.1) About the Author

Author: Anthony Berkow
E-Mail: aberkow@syfrets.co.za
Version: 0.50 (to be updated)

4.2) Basics

The keyboard communicates on a character-by-character basis. Each key, including SHIFT, CTRL and ALT, sends a specific code when depressed and another code when it is released (the break code is the make code preceeded by F0h). ASCII is impractical for communication with a keyboard (especially with special features such as typematic rate) so special scan codes are used and converted to ASCII by the BIOS.

Certain keys produce series of scan codes called scan sets. There are 3 scan sets in use with set 2 being the most common. (Sets 2 & 3 used on AT keyboards.)

The 101/102 keyboard is laid in a 16 row by 8 column matrix. With set 1 or 2, for cursor control keys: ALT, CTRL, DEL, PgUp, PgDn, Ins, Home, End, the keyboard issues a series of codes dependant on shift keys (alt, ctrl, shift) and on the status of the indicator of the Num lock key. Since these keys are duplicated, the basic scan codes are identical - to identify the alternate key, an extra code E0h is added to the basic code. With set 3, each key generates just one code unaffected by the status of other keys.

With set 1, the up code is obtained by adding 80h to the down code. For set 2 and 3, the 1st byte is F0h and the 2nd byte is the down code for that key.

If a key is pressed, the keyboard acknowledges the key and issues the down key code. If a 2nd key is pressed while the 1st is still depressed, the 2nd key is acknowledged and it's down code sent. If the 2nd key is released before the first, the 1st key is deactivated. To reactivate the 1st key, it must first be released.

If two or more keys are pressed simultaneously, all are validated and all codes are sent (no error is generated).

With the exception of the pause key, all keys when held down for a certain time, auto-repeat. The down code is repeated until the key is released. If two or more are pressed together, only the last key pressed is repeated. The repeat stops when the last key pressed is released, even if the other keys are still depressed. The delay is usually 500ms and the repeat 10/s and can be modified.

The interface is serial, using the KBCLK (generated by the keyboard) and KBDTA lines. KBDTA is bidirectional! Data format is 11 bits for scan sets 2 & 3 - 1 start bit, 8 data bits (lsb 1st), 1 odd parity bit and 1 stop bit.

Data from controller to keyboard always has priority. As long as the keyboard has not yet transmitted the 10th bit, the controller can take over the interface, which it does by pulling KBCLK low for 100ms. Within 5ms the keyboard will start sending low clock pulses on KBCLK. KBDTA must be set to the relevant logic level each time KBCLK is low and must remain there until KBCLK goes high then low again. The keyboard reads this data bit when the clock pulse is high. If there is no clock within 20ms or transmission takes more than 2ms assume a transmit time-out and send a resend command to the keyboard (FEh).

For the keyboard to send the controller data, both KBCLK and KBDTA must be high. The keyboard begins by pulling KBDTA low and then starts sending clock pulses on KBCLK. The data bit is output while the clock is high and remains while it goes low then high again. The controller should read the data while the clock is low. If the controller holds the clock low the transmission will be aborted and retried once the line is free.

After the POST (power on self test) the keyboard issues a AAh to the system (FCh if failure).

The signals source is open-collector TTL (low <0.8V high>2.4V). Signals are on a 6-pin DIN or SDL (PS/2) connector (5 wires - 1 supply, 2 gnd, dta and clk):

1. Data In/Out
2. N/C
3. Gnd
4. +5V
5. Clock
6. Test (not used / Gnd)

Chapter 5) Nick Toop's AT/XT keyboard info

5.1) About the Author

Author: Nick Toop
Version: 1.00

I use both XT and AT keyboards with the MC68681 DUART in 68000 based microcomputer systems. I hope the following information proves to be both correct and useful.

5.2) Connector

The connector will be one of the following:

```
5pin 180 deg DIN                6pin MINIDIN
----------------                ------------


      (2)      1 CLOCK            (5) (6)        1 DATA
    (5)  (4)   2 DATA                 |           2
   (3)    (1)  3                 (3)     (4)      3 GND
               4 GND                               4 +5V
               5 5V               (1) (2)          5 CLOCK
                                                   6
```

The signals are open drain and TTL compatible.

5.3) AT Keyboard

The interface is bi-directional. I have not described the various setup options here; they include controlling the LEDs, echoing, changing the key modes, autorepeat and delay parameters, reset. I find the defaults work well enough.

After power up, a successful diagnostic test sends AA (hex).

The idle state is clock and data high. A code is sent when a key is first pressed (and when autorepeat is active). The same code is sent; prefixed by F0 (hex), when the key is released The data bit time is typically 50uS. The timing is as follows:

```
CLOCK
---___---___---___---___---___---___---___---___---___---___---___---

DATA
--_______| D0  | D1  | D2  | D3  | D4  | D5  | D6  | D7  | PO  |------
  START   LSB                                    MSB   ODD   STOP
                                                       PARITY
```

NB: Some keys return an escape code sequence i.e. E014 (Rctrl on) and F0E014 (Rctrl off).

5.4) PC-XT Keyboard

The idle state is clock high and data low. Take the clock low for > 6mS to get diagnostic test. Take data low and issue >62uS clock to lock keyboard.

A code is sent when the the key is first pressed (and when autorepeat is active). The same code is sent with D7 set as the key is released. The pulse widths are in the range 30->50uS. The timing is as follows:

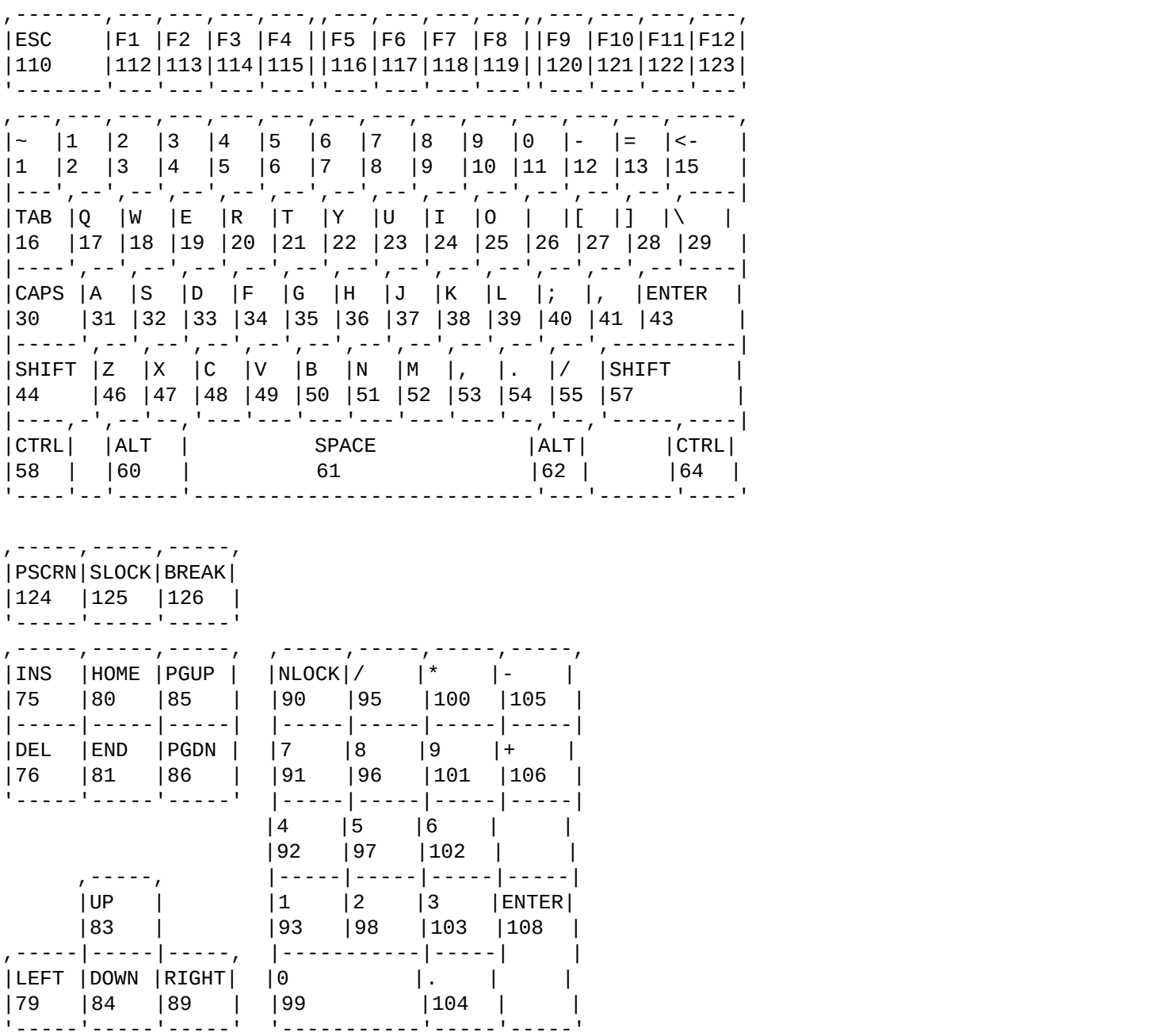
```
CLOCK
---___---___---___---___---___---___---___---___---___---___--------

DATA
____-------| D0  | D1  | D2  | D3  | D4  | D5  | D6  | D7  |-----____
    START    LSB                                    MSB
```

NB: Some keys return an escape code sequence i.e. E014 (Rctrl on) and E094 (Rctrl off).

5.5) Keys to Keycodes

Here are key numbers for a US-English 101 key keyboard.



There are at least three options for the mapping of key numberss to keycodes. The common usage seems to be as follows:

KEY #	PC-XT (HEX)	AT (HEX)	KEY #	PC-XT (HEX)	AT (HEX)	KEY #	PC-XT (HEX)	AT (HEX)	KEY #	PC-XT (HEX)	AT (HEX)
1	29	0E	2	02	16	3	03	1E	4	04	26
5	05	26	6	06	25	7	07	2E	8	08	36
9	09	3D	10	0A	46	11	0B	45	12	0C	4E
13	0D	55	14	--	--	15	0E	66	16	0F	0D

```
|17 |10   |15   ||18 |11   |1D   ||19 |12   |24   ||20 |13   |2D   |
|21 |14   |2C   ||22 |15   |35   ||23 |16   |3C   ||24 |17   |43   |
|25 |18   |44   ||26 |19   |4D   ||27 |1A   |54   ||28 |1B   |5B   |
|29 |2B   |5D   ||30 |3A   |58   ||31 |1E   |1C   ||32 |1F   |1B   |
|33 |20   |23   ||34 |21   |2B   ||35 |22   |34   ||36 |23   |33   |
|37 |24   |3B   ||38 |25   |42   ||39 |26   |4B   ||40 |27   |4C   |
|41 |28   |52   ||42 |--   |--   ||43 |1C   |5A   ||44 |2A   |12   |
|45 |--   |--   ||46 |2C   |1A   ||47 |2D   |22   ||48 |2E   |21   |
|49 |2F   |2A   ||50 |30   |32   ||51 |31   |31   ||52 |32   |3A   |
|53 |33   |41   ||54 |34   |49   ||55 |35   |4A   ||56 |--   |--   |
|57 |36   |59   ||58 |1D   |14   ||59 |--   |--   ||60 |38   |11   |
|61 |39   |29   ||62 |E038 |E011 ||63 |--   |--   ||64 |E01D |E014 |
|65 |--   |--   ||66 |--   |--   ||67 |--   |--   ||68 |--   |--   |
|69 |--   |--   ||70 |--   |--   ||71 |--   |--   ||72 |--   |--   |
|73 |--   |--   ||74 |--   |--   ||75 |E052 |E070 ||76 |E053 |E071 |
|77 |--   |--   ||78 |--   |--   ||79 |E04B |E06B ||80 |E047 |E06C |
|81 |E04F |E069 ||82 |--   |--   ||83 |E048 |E075 ||84 |E050 |E072 |
|85 |E049 |E07D ||86 |E051 |E07A ||87 |--   |--   ||88 |--   |--   |
|89 |E04D |E074 ||90 |45   |77   ||91 |47   |6C   ||92 |4B   |6B   |
|93 |4F   |69   ||94 |--   |--   ||95 |E035 |E04A ||96 |48   |75   |
|97 |4C   |73   ||98 |50   |72   ||99 |52   |70   ||100|57   |7C   |
|101|49   |7D   ||102|4D   |74   ||103|51   |7A   ||104|53   |71   |
|105|4A   |7B   ||106|4E   |79   ||107|--   |--   ||108|E01C |E05A |
|109|--   |--   ||110|01   |76   ||111|--   |--   ||112|3B   |05   |
|113|3C   |06   ||114|3D   |04   ||115|3E   |0C   ||116|3F   |03   |
|117|40   |0B   ||118|41   |83   ||119|42   |0A   ||120|43   |01   |
|121|44   |09   ||122|57   |78   ||123|58   |07   ||124|*1   |*2   |
|125|46   |7E   ||126|*3   |*4   ||127|--   |--   ||128|--   |--   |
'---'-----'-----''---'-----'-----''---'-----'-----''---'-----'-----'

*1 is E02AE037
*2 is E012E07C
*3 is E11D45E19DC5
*4 is E11477E1F014F077
```

Chapter 6) 6502 assembly code for controlling an AT keyboard

6.1) About the Author

Author: Jim Rosemary
E-Mail: jsr@magpage.com

6.2) Introduction

This code was taken from a terminal emulator that I wrote a while back. The entire program was written in machine language and each revision was burned into an EPROM for the target system. Because of this, I did not try to make the code too 'pretty' - I just wanted to get it running. Read at your own risk.

Also, be advised that not all keyboards work with my machine. The better of the two keyboards that I have tried works fine, whereas the cheap one will not work at all.

It is advisable to obtain the keyboard docs from ftp.ee.ualberta.ca in the /pub/cookbook directory and read them. Also, these texts contain similar (if not newer) information.

A 6522 is mapped at $fe30
PB0 goes to DATA
PB1 goes to CLOCK

Also, one of the control lines (CB1 or CB2, whichever corresponds to Interrupt enable $10)

No interrupt code is provided here; only the actual transfer.

$ff is used for passing data to/from the Get Byte and Write Byte routines. Reset has no parameters.

6.4) The Code

```
Get Byte:

                lda $fe32
                and #fc
                sta $fe32
                jsr get_bit
                ldy #$09
loop:           jsr get_bit
                ror $ff
                dey
                bne loop
                jsr get_bit
                lda $ff
                rts

get_bit:
                lda #$02
wait_for_hi      bit $fe30
                bne wait_for_hi
                lda $fe30
                pha
                lda #$02
wait_for_low     bit $fe30
                bne wait_for_low
                pla
                lsr
                rts

Write Byte:
                lda $fe30
                and #$fc
                sta $fe30
                lda $fe32
                ora #03
                sta $fe32
                ldx #$c0
wait:           dex
                bne wait
                lda $fe32            // Write start bit
                and #$fd
                sta $fe32
                lda #$02
wait_for_hi      bit $fe30            // Wait for clock line
                beq wait_for_hi
wait_for_lo      bit $fe30
                bne wait_for_lo
```

```
                ldx #$08
                lda $fe32
                ldy #$01
loop            lsr $ff
                jsr write_bit             // Write data bits
                dex
                bne loop
                tya
                lsr                       // Write parity
                jsr write_bit
                sec                       // Write stop bit
                jsr write_bit
                lda #$01
wait_lo              bit $fe30            // Wait for handshake from kbd
                bne wait_lo
                rts

write_bit
                lda $fe32
                and #$fe
                bcs write_one
                ora #$01
                iny
write_one     sta $fe32
                lda #$02
wait_for_lo      bit $fe30
                bne wait_for_lo
wait_for_hi      bit $fe30
                beq wait_for_hi
                rts

Reset Keyboard:
                lda $fe30
                and #$fc
                sta $fe30
                lda $fe32
                ora #$03
                sta $fe32
                ldx #$c0
wait            dex
                bne wait
                lda $fe32
                and #$fd
                sta $fe32
                lda #$02
wait_for_hi      bit $fe30
                beq wait_for_hi
wait_for_lo      bit $fe30
                bne wait_for_lo
                ldx #$08
                lda $fe32
                and #$fe
                sta $fe32
                lda #$02
wait_for_hi1     bit $fe30
                beq wait_for_hi1
wait_for_lo1     bit $fe30
                bne wait_for_lo1
                dex
                bne wait_for_hi1
                lda #$02
wait_for_hi2     bit $fe30
```

```
                beq wait_for_hi2
wait_for_lo2     bit $fe30
                beq wait_for_lo2
                lda #$01
wait_for_lo3     bit $fe30
                bne wait_for_lo3
                lda #$10
                sta $fe3d
                rts
```
