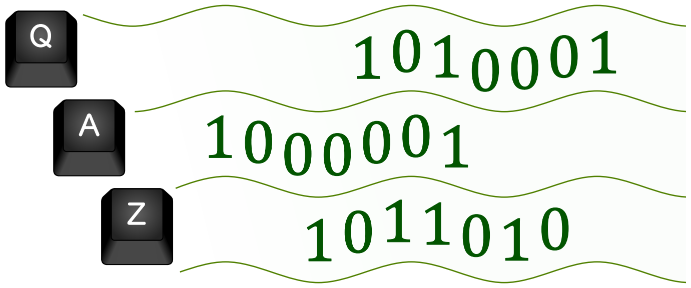
Encoding and output

Contents



Encoding

When a key on a keyboard is pressed, the equipment to which the keyboard is attached needs to know the identity of that key. Thus, some means is required to uniquely identify each key and provide its numeric code. This code may be an arbitrary number (such as model-specific or protocol-specific scancode) or it may be an externally-specified number such as the ASCII code for the key.

There have been numerous solutions to this requirement over the decades. Since the 1980s, this task has commonly been performed by a microcontroller within the keyboard, either on the same PCB as the switches, or on a separate board (the latter being always the case with membrane keyboards). Microprocessors were introduced between 1969 and 1971, so keyboards designed prior to this date could not take advantage of them. Instead, wiring or logic circuitry within the keyboard or encoding within the switches themselves served this function. The first microprocessor-based keyboards appear to have been introduced in 1977.

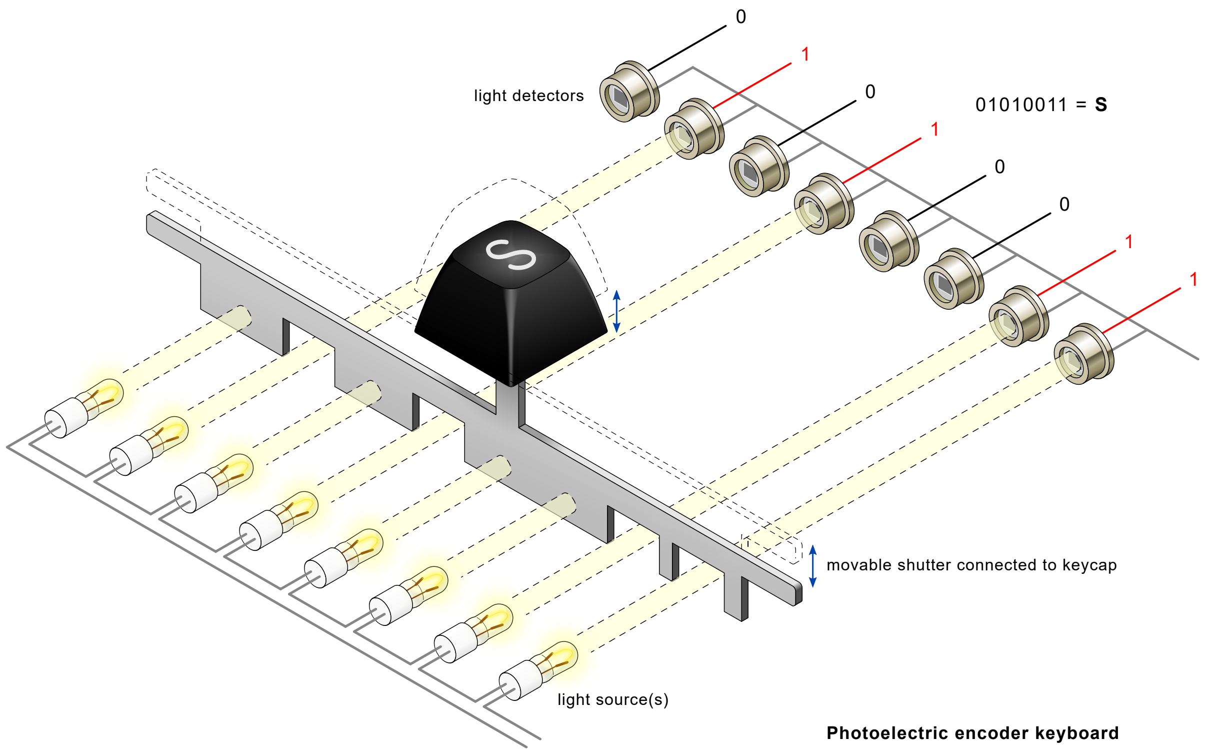
Encoding can be broadly divided into "static" and "scanning" types. So-called "static" techniques are entirely idle until a key is struck; with no oscillator, they do not generate RFI. Some manufacturers such as Stackpole (referring in their case to Stackpole/Magsat keyboards) avoided scanning in part because of the extra cost of dealing with RFI. Scanning keyboards also require a significant amount of logic that one presumes was prohibitively expensive in the 1960s. Around 1970, scanning encoding was introduced; it was more flexible than static encoding (in designs where the key configuration was held in ROM or EPROM), and it allowed the centralisation of the electronics necessary for the growing capacitive keyboard market. Scanning works well with cheap, simple mechanical switches as it makes de-bounce practical, and it allows full N-key rollover with such switches, at a cost of only one diode per key, and practical rollover (what is now called two-key rollover) without diodes.

Encoding keyboards

"Coding" or "encoding" keyboards and switches define the output bits of each key within the key itself. When the key is pressed, the output code is directly generated as a series of parallel bits. There is no need to process matrix co-ordinates or wiring grids: the identity of the key is directly generated from the key itself. Such approaches prevented the need for what was then complicated and expensive electronics, but the encoding process was often bulky, complicated to assemble (every switch needed to be assembled or wired with its dedicated output code), and generally unable to detect simultaneous keypresses (as the keys generally shared a common output bus). As the cost of electronics fell and matrix scanning became affordable, encoding keyboards fell out of favour, allowing much smaller and simpler keyboards to be assembled.

Photoelectric encoder keyboards

Photoelectric encoder keyboards use light beams interrupted by slotted shutters to encode keystrokes. Each light beam encodes a single bit of output. The output codes are defined by the presence or absence of slots or holes for each light beam. The following illustration depicts a generalised idea of how such a keyboard functions:



Photoelectric encoder keyboard

The oldest known such full keyboard was invented by Invac around 1960, and was originally designed to encode a manual typewriter. (Monroe had previously filed US patent 2641753 "Photoelectric keyboard" in 1951 for a photoelectric encoding keyboard for calculators.) Shortly afterwards, they developed a dedicated photoelectric keyboard. Invac's models used a ball bearing tray interlock to physically prevent more than one key being pressed simultaneously, as well as solenoid-driven latch to keep the shutter in place long enough to register the output.

The specific details varied between manufacturers. One documented alternative method (patented by Western Digital: US patent 3818485 "Keyboard apparatus", filed in March 1973) uses a single light source combined with a parabolic reflector to collimate the beam, hence such keyboards being sold by Collimation, Inc. Specifically, the Collimation D40.592 used with the Compucolor 8001 and Intecolor 8001 computers.

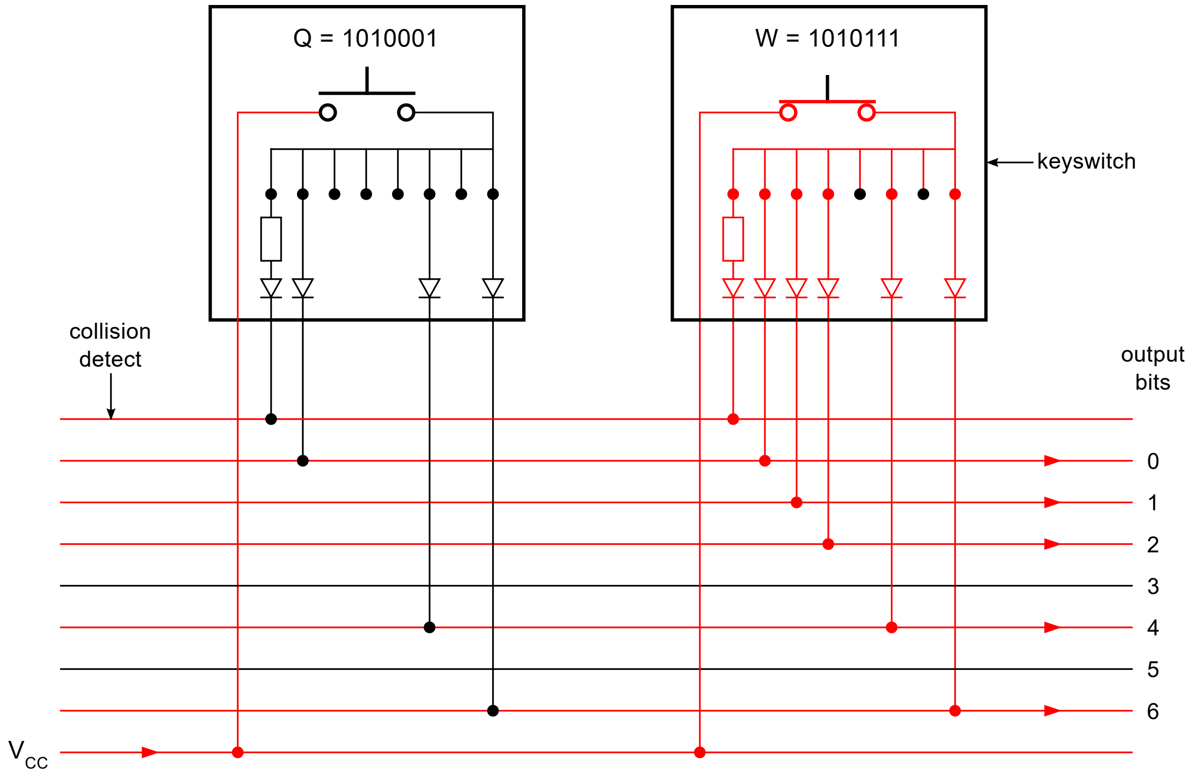
In its basic form, such keyboards do not allow rollover, as placing two shutters across the light beams would result in incorrect output. Jacob Alexander reported that the Collimation D40.592 achieves two-key rollover, although the means by which it does this is not known, and rollover does not appear to be mentioned in the patent.

Viatron's implementation from 1969 used four-of-N code, which allows clashes to be detected.

Encoding switches

Various manufacturers designed self-encoding switches (referred to as “encoding” or “coding” switches). These are discrete switches that directly output the binary code of the character they represent, rather than a complex mechanical assembly such as the aforementioned photoelectric keyboards. As a result, all the switches within such a keyboard sit on a common bus at least eight bits wide: one V_{CC} line to power all the switches, and seven output lines to receive the output bits, such as the ASCII codes of each switch. The switches are supplied by the manufacturer unconfigured, and they are each set to their appropriate code when the keyboard is assembled. This takes the form of clipping or bending terminals, or fitting interconnecting pieces between the switch and each bus line.

The earliest known type is Micro Switch KB, introduced around 1964. These provided 8-bit output. Around 1969, Mechanical Enterprises introduced an encoding keyboard switch model within their Mercutronic family. These provided ten output bits (eleven in the patent), and were solderless, using conductive lines on the back of adhesive tape to form the bus. The diagram below is based on the Mercutronic design, simplified to 7-bit output:



Mechanical Enterprises encoding switch implementation

This approach has a major advantage in that no logic circuitry is needed to identify each key. However, there is also a disadvantage of using a common bus, as rollover is not possible: pressing two more keys simultaneously will jam the bus. Thus, a method is required to determine when two or more keys are pressed at once. Mechanical Enterprises resolved this by detecting the current in a specific bus line. The terminal for this line was fitted with a resistor so that current flowing out of that terminal would be limited. When a second key is pressed concurrently, it also feeds the same bus line, through another resistor of the same value. Two parallel feeds from the same source through identical resistances results in the effective resistance being halved, so the current will double. The higher current would demonstrate to the control circuitry that two or more keys were active at once, and it could then disregard any output until the correct current level was restored (such as when the first key completes its release). This is illustrated in the diagram above.

There is another problem with this approach: current is able to enter an inactive switch through any active bus line, pass through the current distributor, and then re-enter the bus. Thus, even an inactive switch will jam the bus. Two solutions to this problem are known:

- Fit every output terminal with a diode, so that current cannot pass into inactive switches; this was the approach taken by Mechanical Enterprises, and is illustrated above;
- Fit every output terminal with its own switch contacts, so that inactive switches are entirely disconnected; this was the approach taken by Micro Switch and Magsat

The need to physically assign each switch with its own output code does require considerable assembly effort, since every switch needs to be modified individually.

Encoding switches were utilised, because Micro Switch KB encoding switches—seemingly introduced in 1964—were still in production at the end of 1969. However, no equipment has yet been observed containing encoding switches, with the exception of the garish example model in Micro Switch’s brochure.

Capacitive encoding

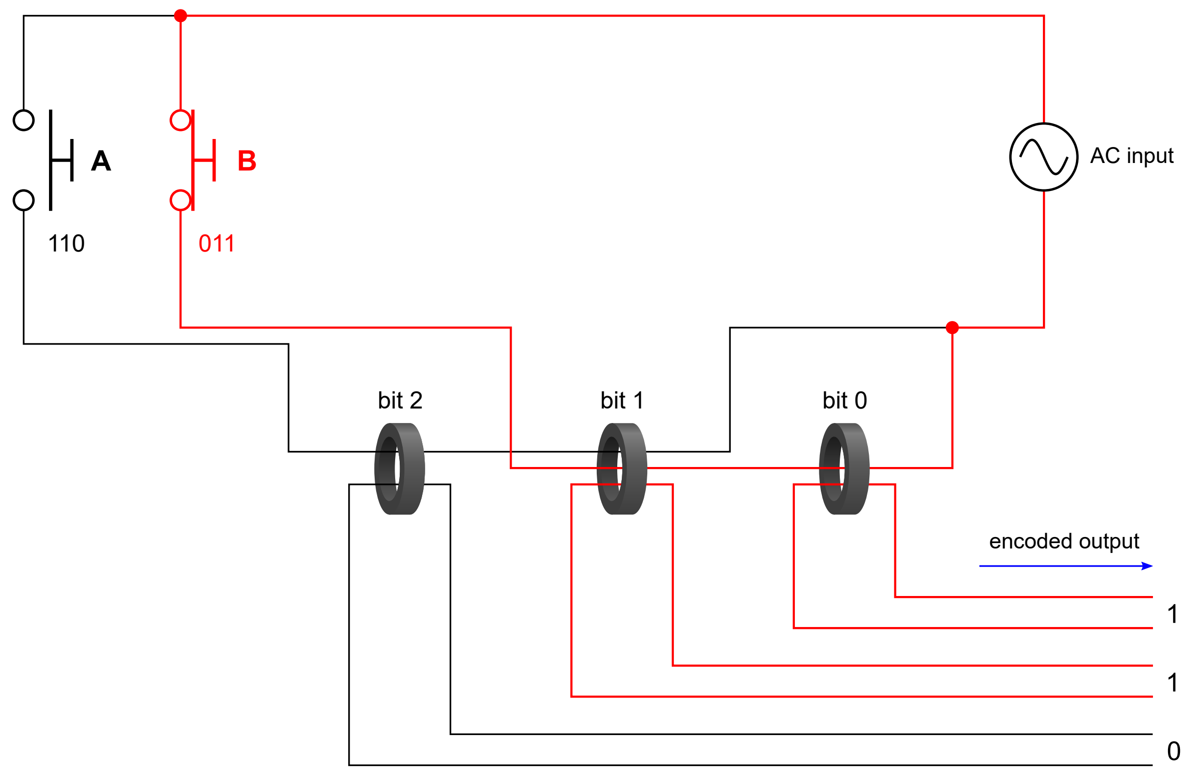
Around 1967, IKOR introduced a capacitive encoding system. Each output bit was represented by a capacitive AC coupling between an aperture on an emitter plate and corresponding PCB track. An electrostatic shield fitted to each key prevented this coupling from taking effect while the key is not pressed. Holes in these shields defined the bit pattern assigned to the key, and pressing the key caused the holes in the shield to line up with the emitter apertures and sensing tracks, allowing the code to be read. This implementation was vertically bulky, even with the apertures and holes placed into two columns, and the design did not seem to last for many years.

Conductive encoding keyboards

Donnelly Mirrors filed patents in 1972 for an encoding system based on formations of tiny contact fingers. The plunger in this design presses down onto an elastomeric sheet that presses in turn into a thin, etched metal layer from which the switch contact fingers are formed. Just as with Micro Switch KB, the strobe connection is made last, this time by altering the finger shape so that the strobe finger is the last one to complete a connection. The logic circuitry handles the rollover conditions, and recreates the lost strobe signal when a clash condition has cleared. Due to the impossibility of depicting this arrangement with mere two-dimensional hatched and outline drawings, it may be that the precise formation of such keyboards will not be known until one is observed; exploded isometric illustrations would have shown the design far more clearly.

Wired inductive encoding

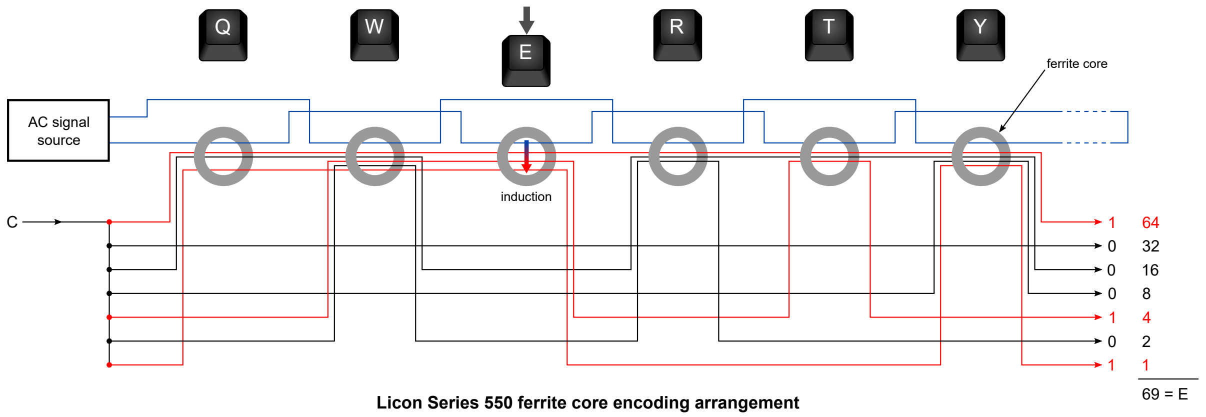
Wired inductive encoding uses ferrite core transformers to generate the output codes. Under actuation conditions, electric pulses fed into the drive line are induced via the ferrite core into the sense lines. Flying leads or PCB tracks are routed through the appropriate cores to define which bits of the output code are 1s. The precise history of this technique is a little unclear. Broadly, there are two approaches to this technique: conductive switching and inductive switching. NAVCOR's model 1067 reed switch keyboards used the conductive option, where the ferrite core arrangement was used solely for encoding; Data Electronics also filed patents on this technique although no further details or examples are known. The diagram below shows a simple example, with two switches and a three-bit output:



Wired inductive encoding with mechanical or reed switches

The second switch (B) is active, and alternating current is passing through it. This induces a current through the ferrite cores of bits 1 and 0, causing those bits to register. It is not possible for current to be induced from the bit 1 core to the bit 2 core via the other switch's circuit (A), because that circuit is open. Thus, only the cores associated with the active switch (B) generate a 1 in the output code. NAVCOR's model 1067 reed switch keyboards routed the switch output through 13 ferrite cores, one per bit. Although only nine cores were needed (eight output bits plus strobe), the drive lines through the ferrite cores were formed from PCB tracks, and space limitations required the upper four bits to be split into two sets of four cores.

The ferrite cores can also form part of the switch itself. Ferrite core sensing uses some means to enable and disable the inductive coupling. The best-known manufacturer of ferrite core sensing is ITW, whose Series 550 keyboards used ferrite cores for both switching and encoding. Series 550 keyboards have a single ferrite core for each switch, and all switches share a set of sense lines, one sense line per output bit. By selectively feeding these sense lines through switches according to the bit patterns required, the encoding of each key was defined through which sense lines were present. The following diagram is derived from the illustration in US patent 3638221 "Solid-state keyboard" filed in 1969, adapted to show full 7-bit ASCII encoding for keys Q–Y:



Licon Series 550 ferrite core encoding arrangement

The blue loop in the diagram above is the signal line, that sends pulses of electricity through the ferrite cores. In the Licon designs, the ferrite cores are suppressed magnetically, with the suppression magnets moved out of the way when the key is depressed. This detail is not depicted for clarity: the diagram shows only the inductive encoding process. Such keyboards were probably only produced for a few years, with Licon moving over to PCB-mounted switches and most likely a matrix scanning technique. PCB-connected switches have been seen as early as 1971.

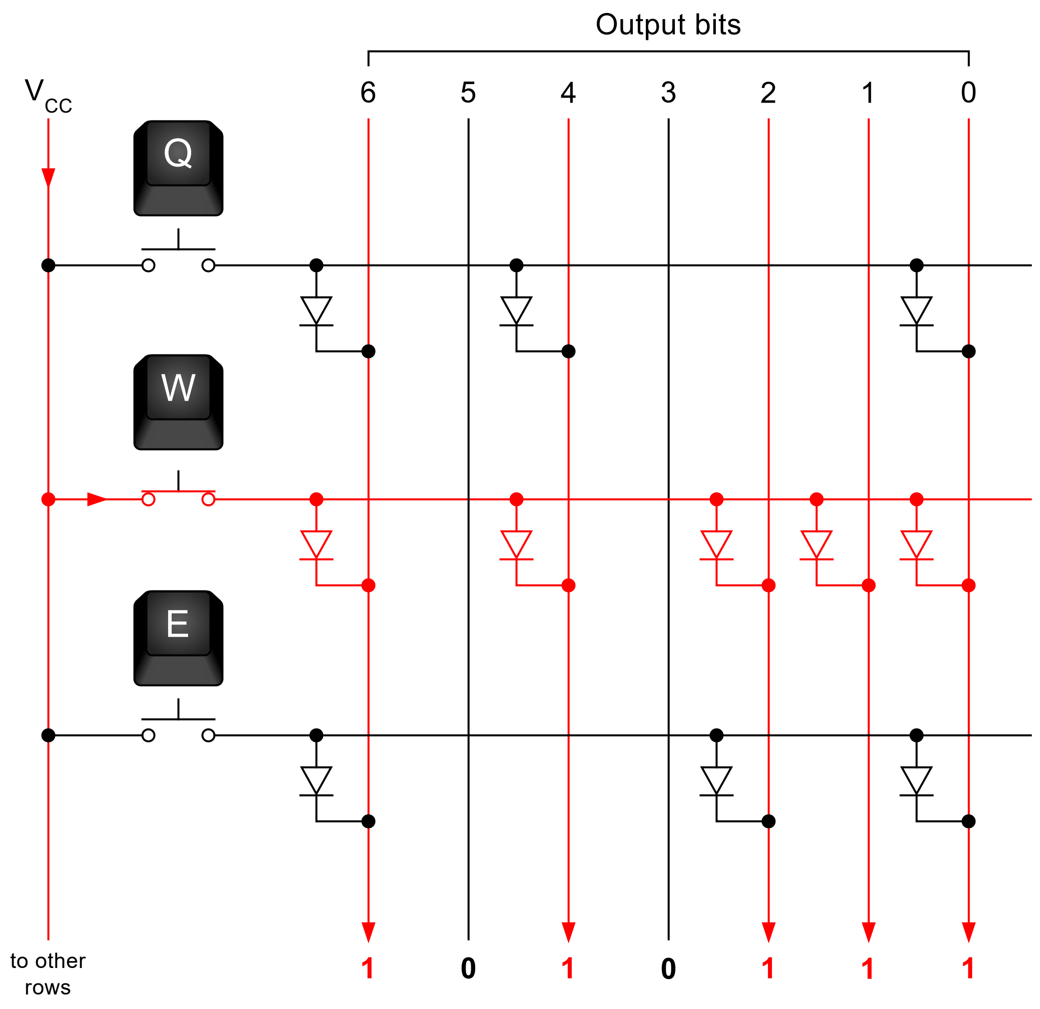
A Micro Switch RW Series 57RW1-2 keyboard has been found with what appears to be a hand-made inductive encoder. Here, flying leads from the switches are routed through seven ferrite cores.

Diode matrix

A diode matrix is a grid of circuit pathways with the row–column intersections connected by diodes. A diode matrix is a precursor to integrated circuit ROM, where each bit is set by the presence or absence of a diode. Instead of using a silicon chip, discrete diodes are soldered to a printed circuit board. Each intersection with a diode passes current from the row into the column, encoding a 1. Where a diode is omitted, no current can pass, and this encodes a 0.

For computer keyboards, diode matrices are a more pragmatic approach to character encoding. They offer logic-free character encoding without the need for a space-consuming parallel bus or special switches. Instead, the diode matrix functions as a lookup table, into which each switch is connected. When a key is depressed, current passes through the switch into a row of the matrix. The switches themselves are not wired into a matrix arrangement: each one has its output wired directly to a row in the diode matrix.

In theory, all the keyboard's logic circuitry then needs to do is read all the columns of the matrix to receive the character code. The following diagram illustrates a diode matrix layout suitable for this purpose:



Diode matrix example 1010111 = W

Diode matrix lookup tables were a successful and commonplace design around 1970, with numerous examples discovered to date. However, most if not all of the examples discovered do not seem to operate in such an obvious manner. The unidentified Micro Switch KB–based example has a straightforward-looking matrix, but the codes defined by the diodes are not ASCII, and appear to be some kind of scancode set. The Potter Instrument matrix is particularly complex. The Scantlin diode matrix may apply simplifications to reduce the diode count, as there seems to be no more than two diodes per key. Tracing the matrices is not easy from photographs because the reverse side of the matrix PCB is often not depicted.

The following examples are known:

- Scantlin Electronics 830 (possibly ca. 1968)
- Potter Instrument KDR/KB 3100A (1970); note that some switches bypass the diode matrix and are routed directly to the edge connector, for direct detection
- Unidentified keyboard with KB switches (ca. 1970): the diode patterns are not ASCII, and the diodes and resistors at the side of the matrix appear to be for resistance-based keystroke collision detection
- George Risk 2-103-001-A-10 (ca. 1970): this model has all the diodes mounted vertically from conductor rails, and also appears to use resistors for collision detection
- UNIVAC 1701 keypunch keyboard (ca. 1971)
- UNIVAC 1710 keyboard (ca. 1972)

The diode matrix implementation was short-lived. A diode matrix consumes a lot of space, either at the back of the keyboard, or on its own circuit board below the switches, and it requires significant additional assembly work. Microchip-based implementations would soon render it obsolete. The diode matrix design retains the limitation of encoding switches in that rollover is impossible. The second key cannot be understood until the first key is released.

Two-of-N code

Two-of-N is a form of constant-weight code where in this instance two and only two bits are always set, and all other bits are cleared. For example, in 2-of-7 code, only the following bit patterns are valid, a total of 21 permissible non-zero values out of the possible 127:

2-of-7 code, possible values

Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
0	0	0	0	0	**1**	**1**
0	0	0	0	**1**	0	**1**
0	0	0	**1**	0	0	**1**
0	0	**1**	0	0	0	**1**
0	**1**	0	0	0	0	**1**
1	0	0	0	0	0	**1**
0	0	0	0	**1**	**1**	0
0	0	0	**1**	0	**1**	0
0	0	**1**	0	0	**1**	0
0	**1**	0	0	0	**1**	0
1	0	0	0	0	**1**	0
0	0	0	**1**	**1**	0	0
0	0	**1**	0	**1**	0	0
0	**1**	0	0	**1**	0	0
1	0	0	0	**1**	0	0
0	0	**1**	**1**	0	0	0

0	1	0	1	0	0	0
1	0	0	1	0	0	0
0	1	1	0	0	0	0
1	0	1	0	0	0	0
1	1	0	0	0	0	0

Two-of-N codes serve as an intermediate representation of key identity, between the switch and the final output code. Additional logic is required to convert between the two-of-N codes and the final output codes, such as ASCII or EBCDIC.

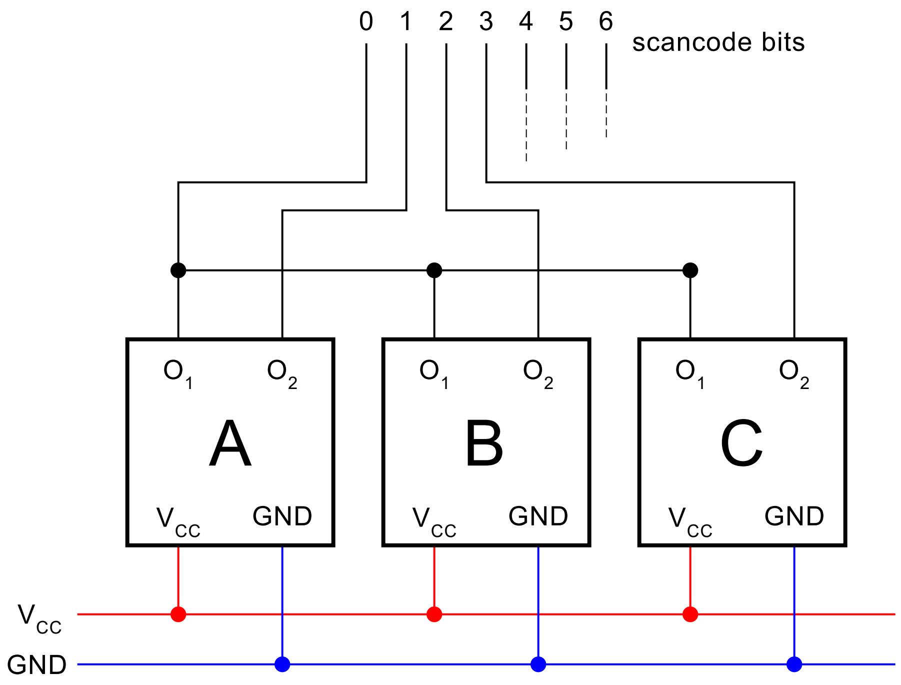
Few keyboards are confirmed to use two-of-N codes; chief amongst these are Micro Switch's Hall effect keyboards, in particular SW Series, but it is likely that it was also used with their RW Series reed keyboards. The keyboard made by Bendix for NASA is almost certainly two-of-N, from the circuit layout, complement of TTL and the double-pole switches. Zbrojovka Brno Consul keyboards made with TESLA contactless keyboard encoders are also two-of-N, confirmed from a circuit diagram supplied with one such keyboard.

National Semiconductor's MM5745 and MM5746 single-chip encoders both support 2-of-13 encoding, but a keyboard using either chip has yet to be discovered. The Electronic Design article *Focus on Keyboards* (vol. 20 no. 23, 9th November 1972, [ED1972-FOK]) depicts two models of Maxi-Switch keyboard, one MOS-encoded and the other TTL-encoded; the MOS-encoded model has a pattern of PCB tracks that appears to be a two-of-N arrangement. The switch type used in the Maxi-Switch keyboards is not explained, but from their cylindrical shape, they are likely to be 2700 series reed switches, which were also advertised in 1972. No mention of 2700 series supporting double-pole operation is known, but it was common for reed keyboard switches to offer that configuration, making them suitable for two-of-N encoding.

In two-of-N keyboards, each switch is wired with two output paths, one for each of the two active bits in that key's intermediate code. Mechanical and reed keyboards would need either a diode for each output or double pole switches to prevent wrong-direction current flow. However, dual output Hall effect sensors are perfectly suited to driving a two-of-N encoder and it is likely that that this was a deciding factor in providing SW sensors with dual outputs. In Micro Switch's Solid State Keyboards brochure from 1973, they noted:

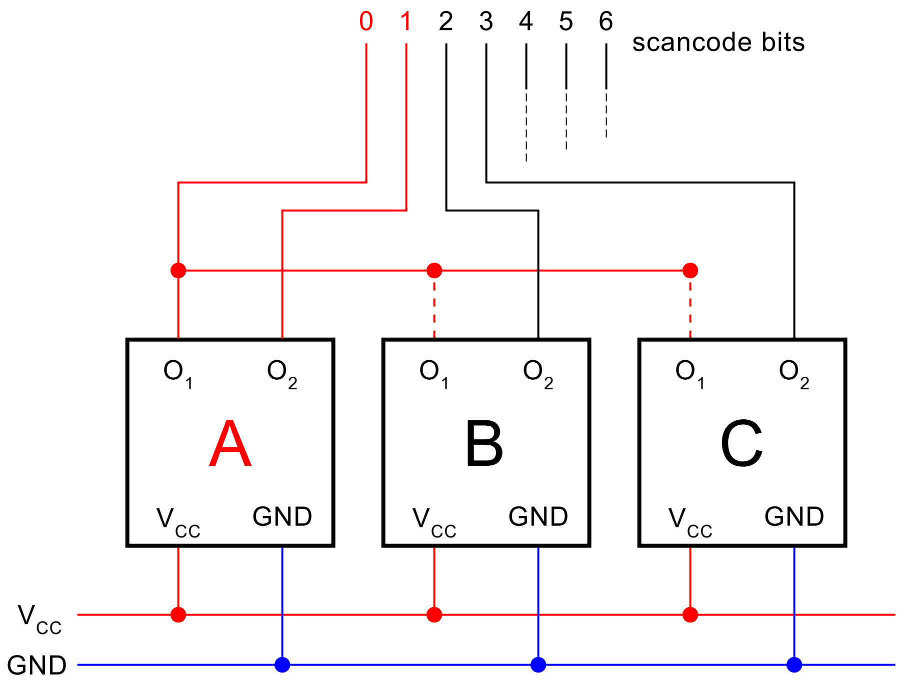
> A unique two-of-n code is developed from the two isolated outputs available from each key switch. This code is used to address the keyboard encoder, thus eliminating the need for complex and costly scanning techniques.

The diagram below shows a portion of a Hall effect keyboard with dual-output sensors, wired in a two-of-N arrangement:

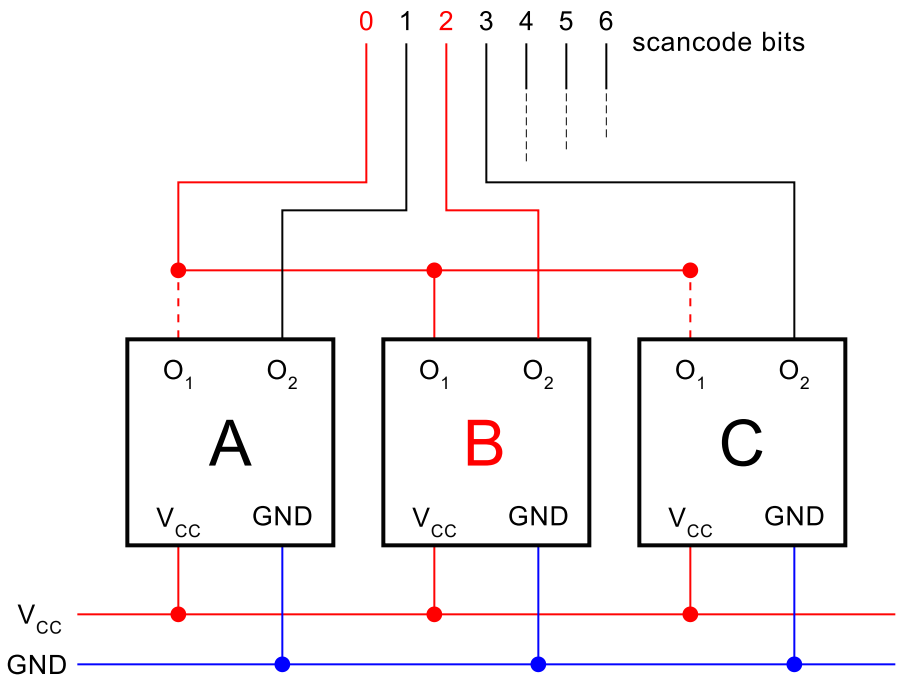


2-of-N: no switches pressed
output = 0000000

You may notice that all the switches are powered up at all times. This approach is simple but it does require considerably more power than a matrix scan keyboard. When a key is pressed, its two outputs enable two bits of data input into the encoder:



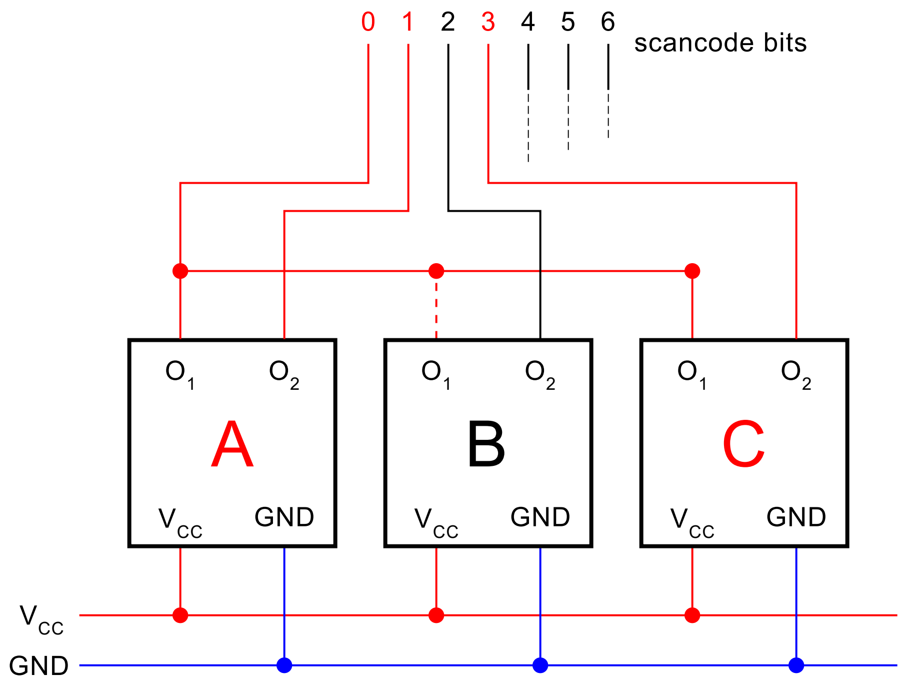
2-of-N: switch A pressed
output = 0000011



2-of-N: switch B pressed
output = 0000101

These intermediate codes are very sparse, and require more bit lines to represent the range of keys present that regular scancodes or ASCII codes.

When two keys are pressed simultaneously, the input into the encoder is illegal:



2-of-N: switches A and C pressed
output = 0001011

Here, the two-of-N code has three digits set, which is not permissible. As such, the keyboard circuit itself has no rollover capability. With switches that remain active while pressed, the encoder needs a way to detect such collisions. Micro Switch keyboards with DTL or TTL encoding logic require an electrical monitor circuit to detect collisions, while MOS-encoded and intelligent keyboards can simply observe that too many bits are set. Under collision conditions, keyboard output is suspended until only one key remains pressed. During rollover conditions, so long as the first key is released before the second, the keyboard will still output the keys in the correct order. Micro Switch also solved the clash problem of direct encoding by producing switches that only register for a tiny fraction of a second. By the time that the operator has pressed the next key, the previous key will have shut off its output even if it remains held. Although this can be achieved with a mechanical switch (as with Micro Switch PB Series or Cherry's equivalent) this ability is much better suited to IC-based switches. Modifier keys such as shift do require switches that continuously generate output while held.

The number of keys available for *N* output lines is the sum of the sequence 0+1+2+3+… up to the *N*th term. Some possible values are given in the table below:

Keys per *N*

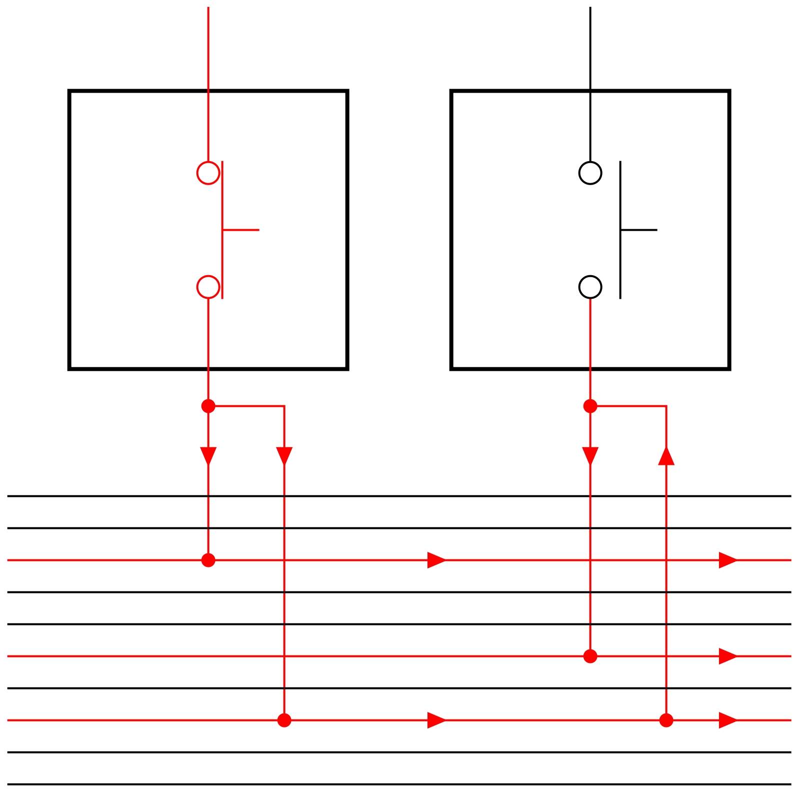
output lines

Lines	Max. keys
2	1
5	10
11	55
12	66
13	78
14	91
15	105
16	120
32	496

With Micro Switch keyboards, *N* in the range of 11–16 are viable for MOS-encoded keyboards with a ROM look-up table. For DTL or TTL-encoded keyboards with 7 or 8-input binary encoders, a full 32 lines is required for 8-bit output codes, as any two-of-N code with both active lines pointed to the same encoder IC is illegal.

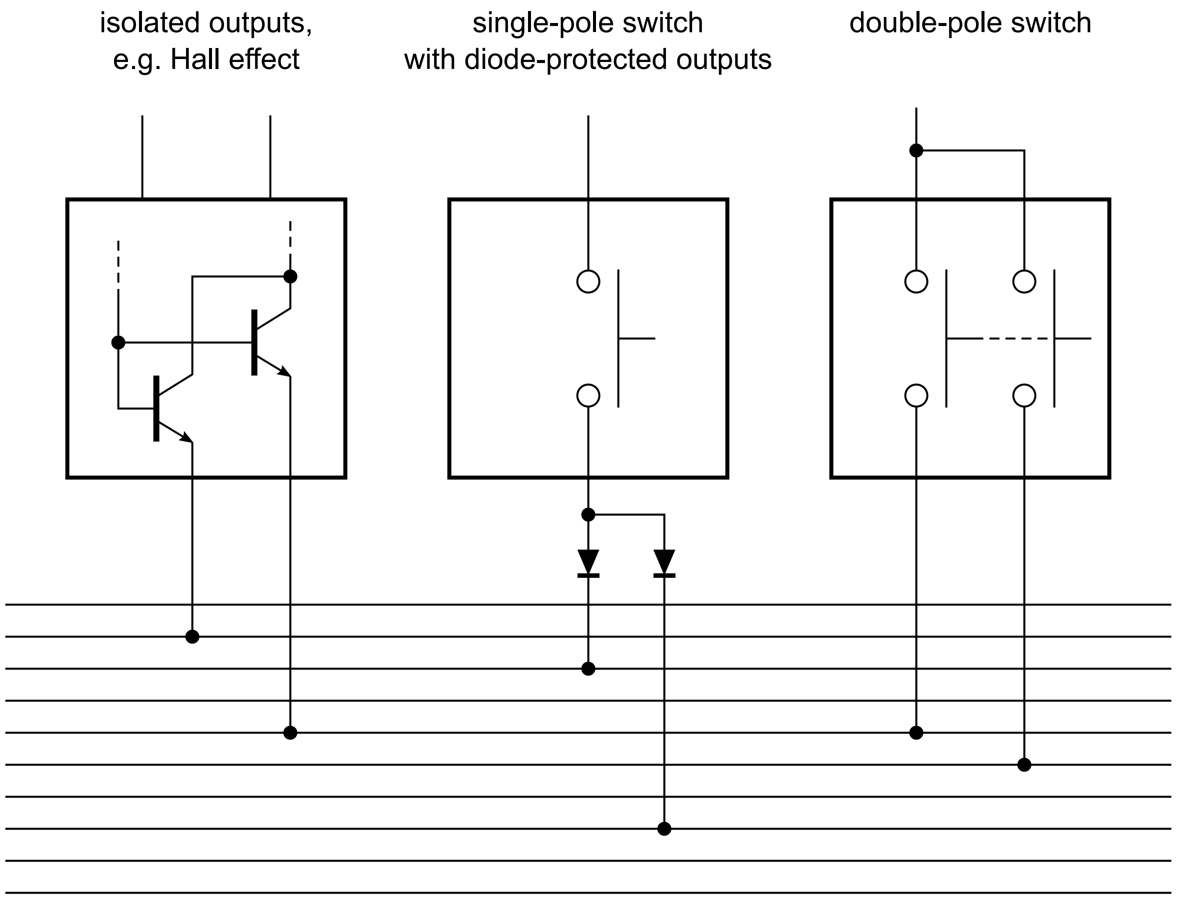
Suitable switch configurations

Used with simple single-pole switches, two-of-N exhibits the same problem as with encoding keyboards, in that current will flow towards inactive switches and back out to unintended output lines:



Wrong current flow, two-of-N

With far fewer outputs than encoding switches, this is relatively easy to rectify. Hall effect switches are perfectly suited to two-of-N arrangements, as those with dual isolated outputs will not pass incoming current through their output transistors. Single-pole switches whose output branches on the far side of the switch contacts need a diode protecting each output. Although NAVCOR's reed switches are only known to have been used with diode matrices, they have been observed with dual diode-protected outputs that would make them suitable for two-of-N encoding. Double-pole switches can be used instead, as these will not pass current between poles.


isolated outputs, e.g. Hall effect
single-pole switch with diode-protected outputs
double-pole switch

Suitable switch arrangements for two-of-N encoding

Magsat's "flying magnet" switches were, in their standard design, fitted with a single common terminal and one or more output terminals. This made them suitable for both self-encoding (if fitted with a full nine contact sets: eight form A plus common) or two-of-N encoding (with the standard two form A model).

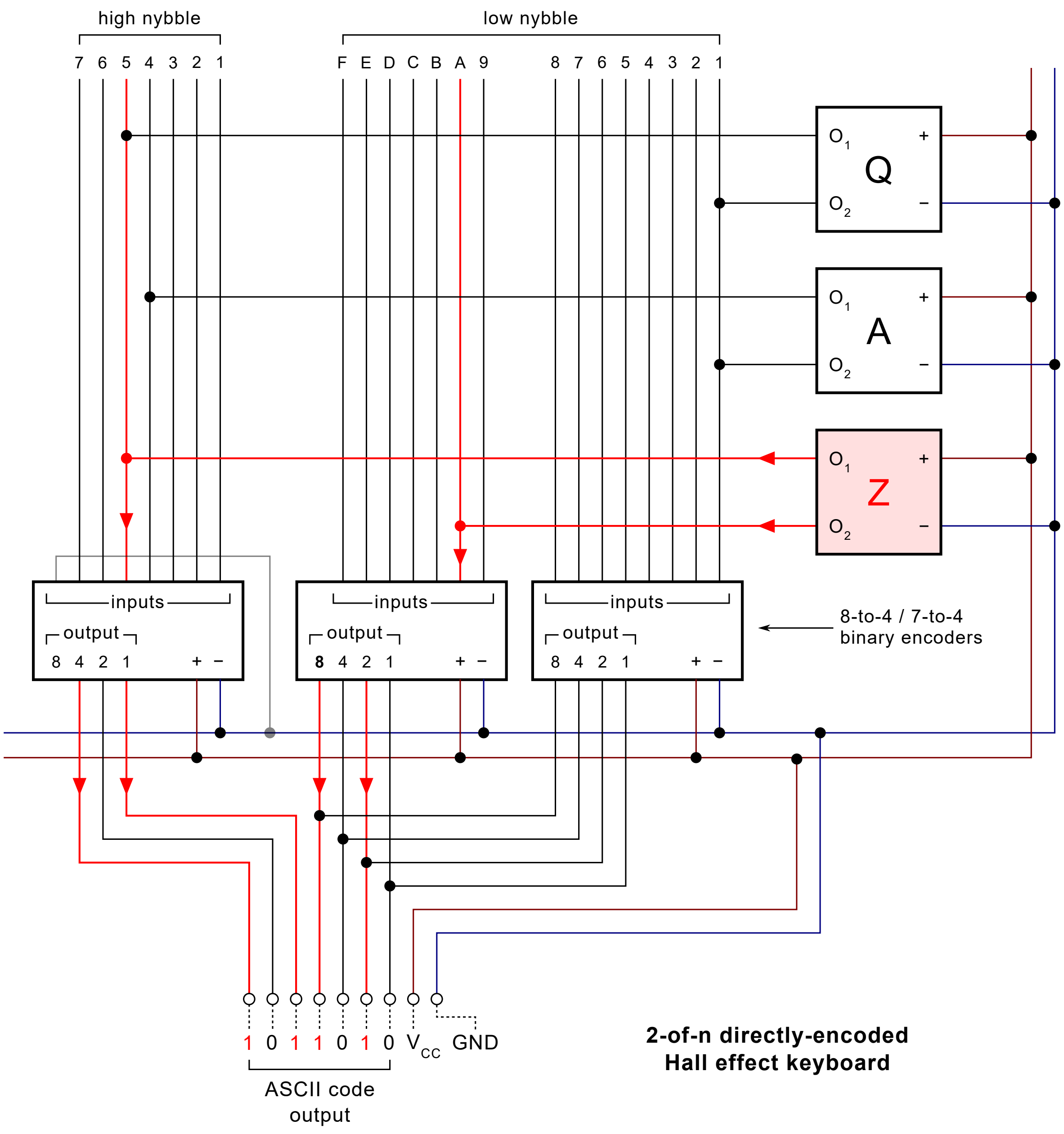
Output code conversion

There are different options for converting two-of-N codes into the final output codes, such as ASCII. Micro Switch noted in 1973:

> We offer a great deal of flexibility in providing codes that meet your exact needs. With MOS (large scale integration), a variety of electronic options is available which can simplify your system as well as increase its versatility. Where encoding and system requirements are less complex, TTL (medium scale integration) encoding is used. Our encoding techniques can generate any code that you specify and in the number of modes necessary for your system. Totally unrelated codes may be generated by the same keys. For instance, a keyboard may be specified that generates two modes of USASCII and two modes of EBCDIC. Since we do not have to logically pair our encoding for multi-mode operation, you may pair characters in any way you choose.

> And for quick turnaround on prototypes and low volume orders we have programmable ROMs. So you can have your exact code required without long delays for new encoded designs. In other words, with our encoding techniques there are almost no limitations.

An example of the TTL encoding arrangement is given in the Mini Bee Computer Terminal Service Manual from 1974 (PDF pages 75–78). Here, 8-to-4-line and 7-to-4-line decimal encoders are used to produce the high and low nybbles for each character from a coding grid similar to a diode matrix. The general principle involved is illustrated in the following highly simplified diagram:



The output code is divided up into its high and low nybbles, and each non-zero nybble value (0x1 to 0x7 or 0xF) is given a separate column on a PCB. One output from each Hall sensor is connected to a column representing a high nybble value, and the other output is connected to a column representing a low nybble value. Groups of columns are fed into binary encoders that convert the index of the input into its binary representation. The encoder outputs are then placed side-by-side to form the complete output code. This way, the output value of each switch is set directly in the circuit wiring, in a manner similar to that of the diode matrix. Because many of the two-of-N codes are illegal here (where both active output lines are connected to the same encoder chip) a 7-bit keyboard requires 22 output lines, and an 8-bit keyboard requires 30. The encoding grid itself is almost as large as the whole keyboard, and necessitated the switches having a PCB of their own, connected all the way along one edge to the encoding board.

The circuit diagrams for the Mini Bee keyboard demonstrate that, in reality, the circuit is more complicated. For example, when the control key is held on an ASCII keyboard, bits 5 and 6 must be suppressed, while with shift held bit 4 must be cleared for number and symbol keys and bit 5 must be cleared for letter keys if the keyboard supports separate uppercase and lowercase. The encoder for the high values (9–15) of the nybbles must also set the highest bit when an input is active. The Mini Bee keyboard also uses an electrical monitor to detect clashes instead of pulse switches to prevent them. Additional circuitry will be present to handle the Repeat key used to request auto-repeat. Alternatively, this functionality could all be handled in software by giving the modifier keys their own dedicated lines on the PCB edge connector, allowing the operating system or ROM monitor the ability to make the adjustments. As noted later, microcomputers moved all this processing to the operating system, where code is cheaper than keyboard control hardware.

The diagram above shows that each switch is connected to two IC inputs using a grid. This is achieved using a double-sided PCB, with the grid columns on side and the grid rows on the other, with interconnects to assign each sensor output to a value. Switch outputs are fed into one axis of the grid, and the other axis is fed into the encoding circuit. This grid arrangement is visible in most of the keyboards in Micro Switch's 1973 brochure. The use of a generic grid on the PCB allows the character encoding to be changed completely without needing to redesign the PCB or exchange any of the circuit components: the entire character encoding is defined solely in the interconnect pattern set in the grid. Detailed photographs of a keyboard using a very similar (if not identical) circuit to that of the Mini Bee keyboard can be seen in Micro Switch keyboard model 64SW1-4. The difference between the 64SW1-4 circuit and that of the Mini Bee keyboard appears to be minimal at most (although they use different PCBs); both use an electrical monitor circuit to detect keystroke clashes, and TTL binary encoding using RW-10038 and RW-10039 ICs, fabricated in the case of 64SW1-4 by Motorola as custom Micro Switch parts. The PCB used for the Mini Bee keyboard and the standard teleprinter keyboard model 53SW1-2 does however use a customised wiring arrangement without a generic grid.

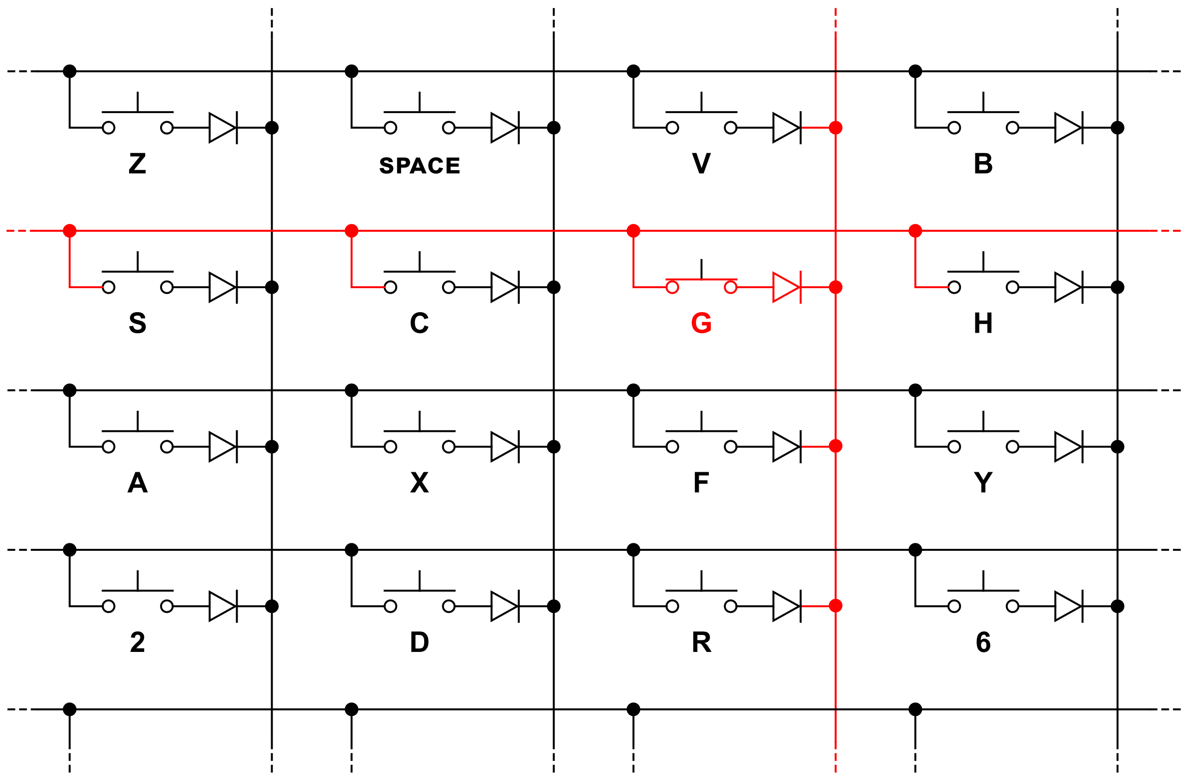
Micro Switch soon introduced MOS-encoded keyboards, using a single-chip LSI encoder. This single chip replaced the electrical monitor and could also replace the entire coding grid, simplifying the circuitry considerably. By placing the intermediate-to-final code conversion into a ROM-based look-up table inside the encoder, a single PCB could provide any number of key arrangements with a simple change of ROM mask. As such, a single output bit trace could branch out to all switches sharing that value: there was no need for each switch to have fully-independent wiring as it would with a coding grid. The two-of-N codes produced by each key were likely sequential: the code for W could be the next one in sequence after the code for W. There were no longer illegal values as there were with the DTL/TTL-based encoding. This simplified the circuit design even further and allowed for a much smaller PCB. The PCB layout for a MOS-encoded SW Series keyboard with PCB code SW-11234 can be found on the MMcM/parallel-ascii-kbd Wiki on GitHub.

Other codes

Viatron used 4-of-8 encoding with the photoelectric encoder keyboards. This is a denser code that is better suited to a light beam keyboard.

Matrix scan

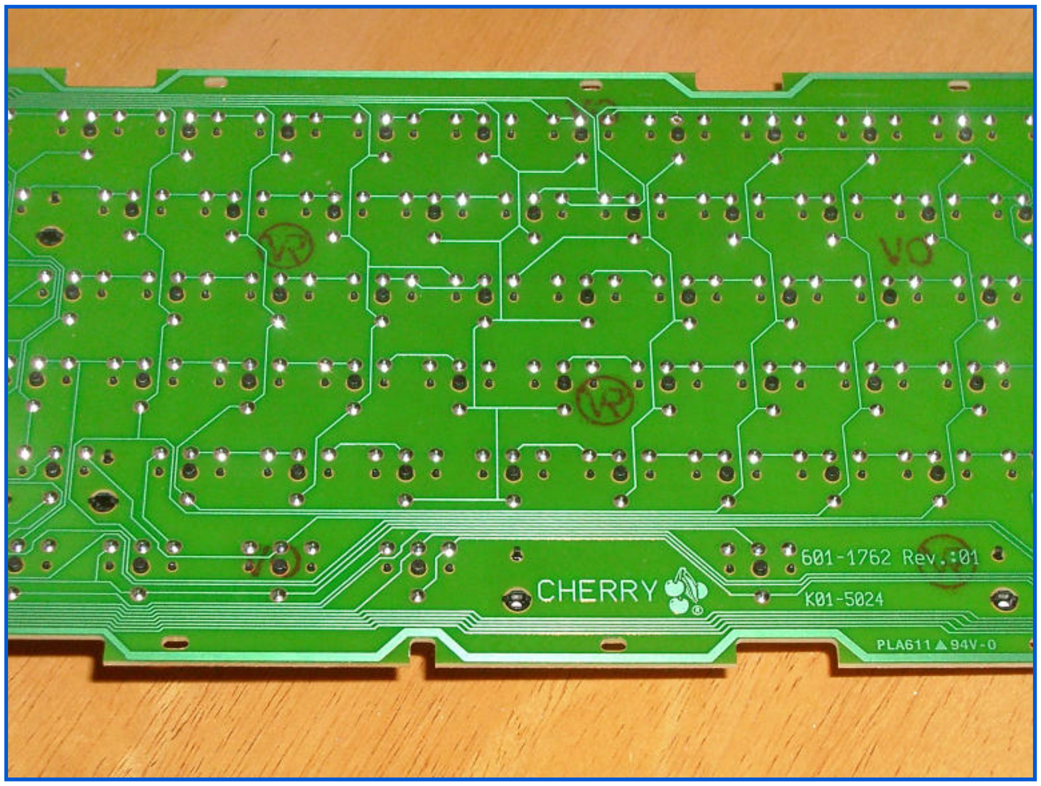
Since the early 1970s or so, the standard way of wiring switches in a keyboard is a matrix. This is a grid of wires—typically PCB tracks—with each key placed at an intersection between a row wire and a column wire. Each key is thus identified by a pair of coordinates (*x*, *y*). Actuating a key causes that position to show as active, whether by electrical conduction or other means of sensing. Circuitry in the keyboard scans each row of the matrix in turn, looking to see which columns are active. The following illustration shows a portion of a switch matrix wired with mechanical contact switches, with diodes fitted to ensure N-key rollover:



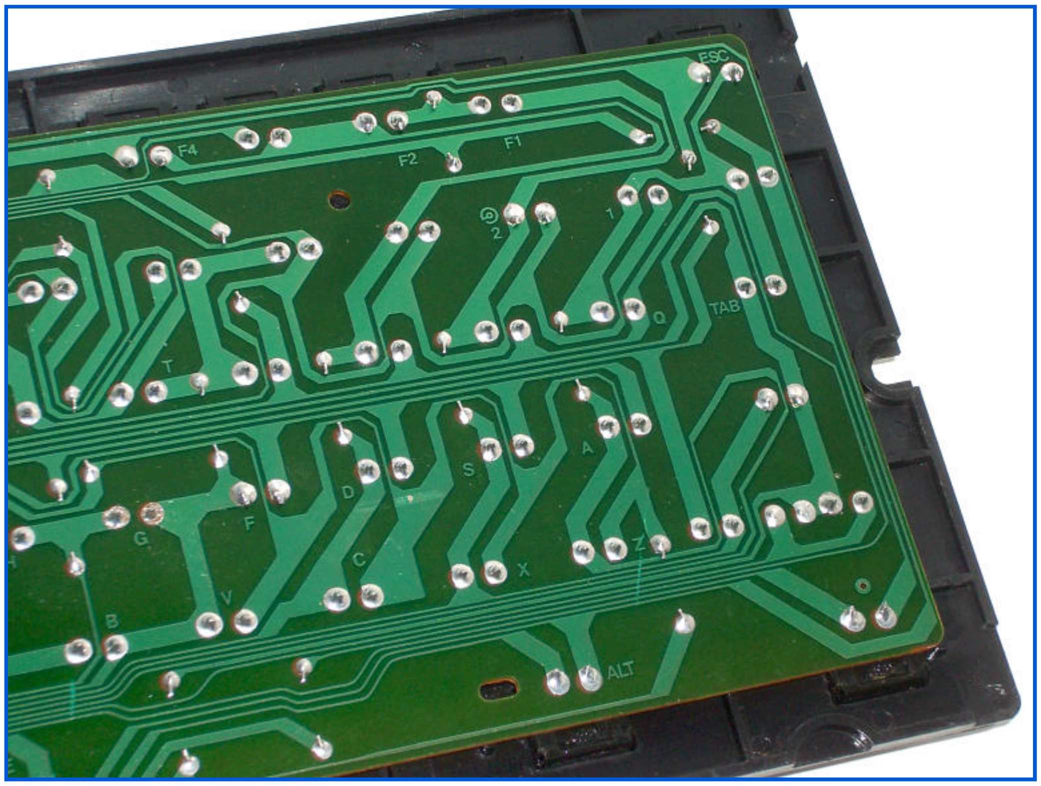
Switch matrix example with N-key rollover diodes

In the diagram above, you will see that the matrix layout does not correspond to the key layout on the keyboard. (The layout of the matrix example above is taken from that of the BBC Micro, but that keyboard only has diodes fitted to one column.) There are several reasons for this, not least the fact that the layout of a keyboard is not a regular grid to begin with. PCB routing and rollover requirements often lead to the PCB tracks being laid out very differently to the logical matrix form that the encoding circuitry sees, although some implementations map the matrix coordinates directly to ASCII codes. The PCB track layout varies widely between manufacturers in terms of how closely it resembles the circuit diagram for the matrix.

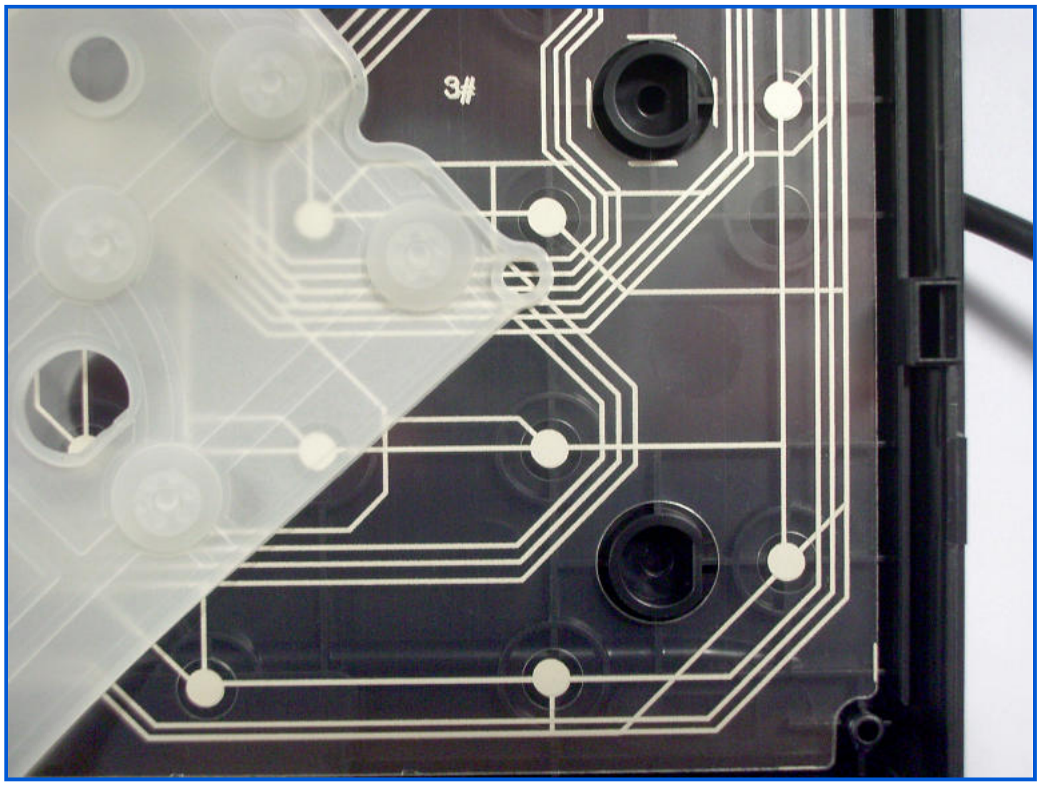
The images below show the matrix layouts of a number of different keyboard models, both mechanical and membrane.



Part of the Cherry G84-4400 matrix; the integrated jumpers in ML switches allow the routing to be very clean, giving a fairly regular track layour



Part of the NTC KB-6251EA matrix, with a much more complex track structure



Part of the Silitek-made Dell KB216 matrix: the matrix arrangement is also used with rubber dome over membrane keyboards

Matrix scan solves a number of problems with earlier keyboard implementations by allowing the sensing and encoding electronics to be centralised (factored) from the switches themselves. Circuitry repeated key-by-key only needs to exist once, in the encoder, which simplifies keyboard implementations and reduces production costs. Matrix scan makes reliable mechanical switching viable by allowing debounce logic to be applied to any key; the simplest approach to debounce is to pause the scanning process to allow time for the switch contacts to settle. Matrix scan also greatly simplifies capacitive keyboards by centralising the capacitive sensing and reducing each key to a simple variable capacitor. Further, as every key has a unique coordinate pair within the matrix, it can be addressed separately, allowing for limitless keys to be held simultaneously without collisions: matrix-based keyboards use the matrix coordinates to identify keys, instead of common bus wiring.

From the inception of matrix scanning around 1970, both discrete logic (DTL and TTL) and large-scale integration (LSI) encoders were used. DTL and TTL encoders use the key positions within the matrix to define the output codes, necessitating a redesign of the PCB to change the identity of keys. A DTL/TTL encoder would use around 12–16 discrete chips. The alternative method was the single-chip encoder, implemented as LSI in MOS circuitry; single-chip encoding arrived around the same time as matrix scan, around 1969. LSI (MOS) encoders provide a look-up table in ROM (or externally in PROM/EPROM) to abstract the output codes of the keys from the matrix positions. For example, changing the first key on the number row from QWERTY (1/!) to AZERTY (&/1) would require a circuit redesign in both the matrix wiring and shift logic in a DTL/TTL keyboard, while in an LSI-encoded keyboard only the look-up table entry would need to be changed. By the late 1970s it became practical to implement encoders as microcontroller firmware.

In modern keyboards, the matrix positions do not provide the ASCII codes of the keys, but the co-ordinates can be used to create internal identities that can be converted to ASCII codes or standard scancodes using a look-up table in ROM.

The matrix arrangement has the significant advantage of allowing two or more keys to be held simultaneously. Because each key position can be examined independently, the state of each key is not inherently affected by the state of any other key. This allows reliable interleaved press and release detection, from two or more overlapping keystrokes, all the way up to an unlimited number of keys held down: rapid typing and holding keys concurrenly does not inherently jam the key detection process.

The limit on the number of keys that can be held without interfering with the scanning process does depend on the type of keyboard in question. Electromagnetic sensing, including ferrite core and capacitive keyboards, provides isolated key examination by nature. Conductive keyboards (such as mechanical keyboards) require a diode at each matrix position to prevent wrong-direction current flow from causing “ghost” or “phantom” keypresses to be detected where no key was actually active; without these diodes present, the number of keys that can be detected concurrently depends on which keys are involved, but a minimum of two can be guaranteed. Unlimited concurrent key detection is the current definition of N-key rollover. Although N-key rollover on a conductive keyboard does still require diodes, only one diode is required per switch, and for ease of assembly these diodes were sometimes fitted into switch in the factory (as is, or was, the case with RAFI, Cherry, Siemens, Clare and others). In most cases, membrane keyboards cannot provide unlimited concurrent keys, as there is no way to solder diodes onto plastic sheets; workarounds do exist, but little is known about most of them. However, N-key rollover is not a necessity and keyboards function fine with a limit of two concurrent non-modifier keys. (Modifier keys are wired specially to avoid problems.)

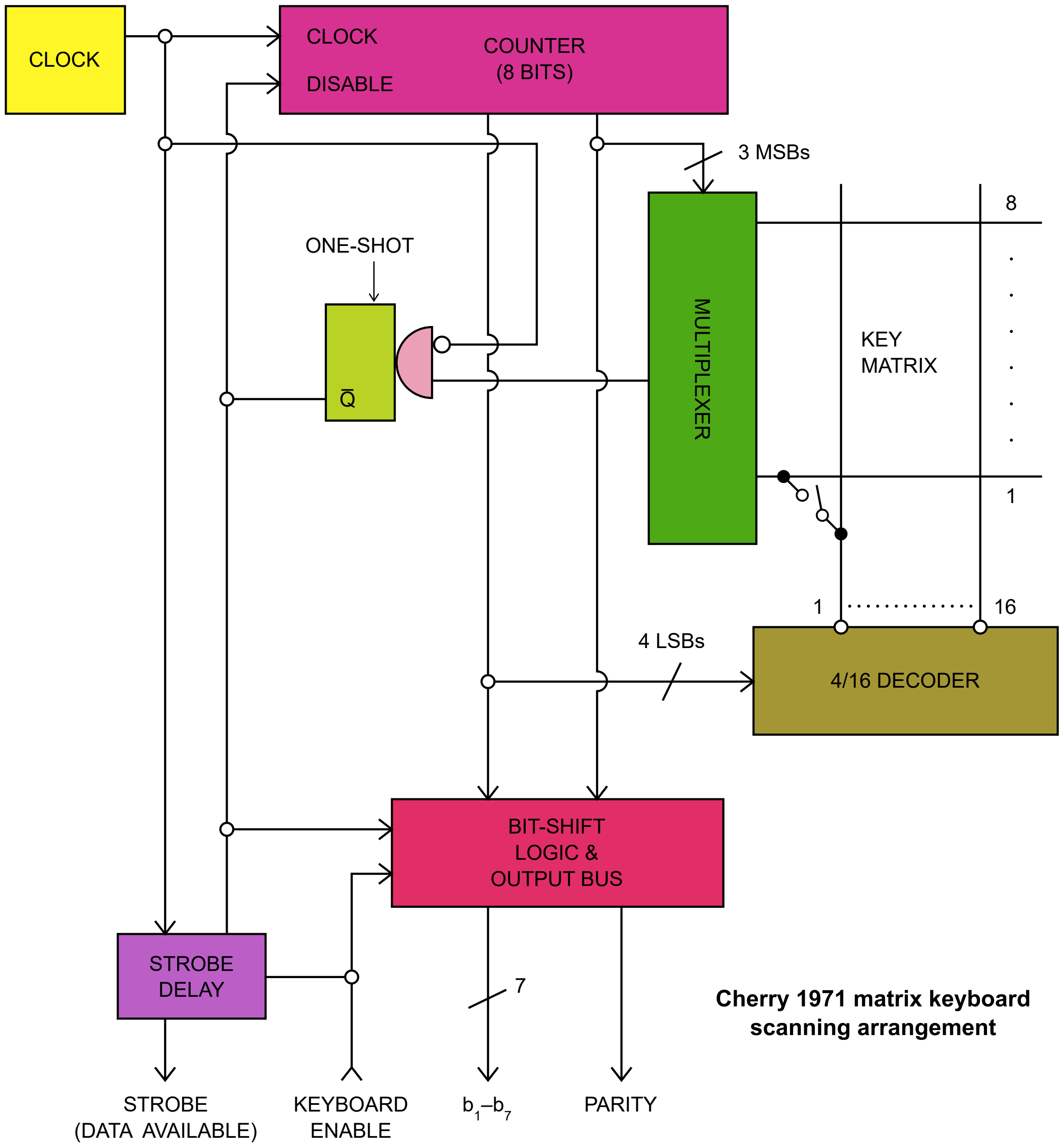
Several companies are known to have advertised or produced matrix scan keyboards in 1970. These include Cherry’s reed keyboards (TTL), Datanetics 01-02-638 (TTL) and Clare-Pendar reed keyboards (LSI). Datanetics also produced an LSI-encoded keyboard in 1971 that one presumes is also matrix scan, although the functionality is not known.

Matrix scan was not a given at this time; Micro Switch continued using their two-of-N static encoding, and the Bendix keyboard produced for NASA around 1974 also used two-of-N static encoding. Photoelectric encoder keyboards would still remain in production for some time to come.

TTL-based matrix encoding

Some keyboards provided the matrix encoding entirely in transistor–transistor logic (TTL), without a dedicated encoder or controller chip. The output code from each key was derived from the key’s matrix co-ordinates. A binary counter IC addresses a multiplexer IC in order to iterate over the matrix rows, and a decoder IC or similar circuit detects the active columns. Where two or more keys are pressed at once, a single key can be filtered using a priority encoder IC, a portion of the counter value (or a second counter), or equivalent logic in discrete TTL chips. The value of the counter plus the value of the active column are combined to form a single binary number representing the output code, that is then presented to the host. The output can be parallel, or fed into a shift register for serial transmission.

TTL-based encoding has the advantage of using entirely off-the-shelf components to form the encoder. This was the basis of Cherry's keyboard designs described in New Cherry electronic data entry keyboards from 1971 (document scanned by Bitsavers). Combined with simple mechanical switches, this likely allowed Cherry to produce an affordable but reliable product. Cherry's block diagram in their brochure was depicted as follows:



Cherry 1971 matrix keyboard scanning arrangement

Cherry's "PRO" keyboard (B70-05AB) utilised fourteen to sixteen integrated circuits, the part numbers of which are given in the documentation:

Part number	Function	Quantity
7400/MC3000	Quad 2-input NAND gate	1
MC846	Quad 2-input NAND gate	1
7402	Quad 2-input NOR gate	1
7408	Quad 2-input AND gate	3
7474	Dual positive-edge-triggered D-type flip-flops	2
7486	Quad 2-input XOR gate	1
7493	4-bit binary counter	2
74123	Dual retriggerable monostable multivibrator	1
74150	16-line to 1-line data selector/multiplexer	1
74154	4-line to 16-line decoder/demultiplexer, inverting outputs	1
74123	Dual retriggerable monostable multivibrator (optional)	1
74180	9-bit odd/even parity generator/checker (optional)	1

A second 74123 multivibrator is one of the optional components that together provide auto-repeat using the repeat key; in addition to the 74123, three extra resistors and capacitors are required, as well as PCB alterations (cutting of a jumper and connecting two locations in the circuit). The 74180 parity generator is the other optional chip, which when installed provides a parity bit in the output; this feature, too, requires alterations to the PCB. Various other changes are possible in the "PRO" keyboard with circuit alterations or changes to resistor and capacitor values, such as timing alterations and changes to the key assignments. The "PRO" keyboard was designed with considerable flexibility, but only by way of altering the circuitry.

The term "bit-shift logic" in the diagram does not refer to the use of a shift register to convert between parallel and serial data (and no such chip exists in the B70-05AB parts list). Rather, it refers to the process of transforming ASCII codes when the control or shift key is held, which is achieved by bit manipulation.

TTL-based encoding also has some significant disadvantages. Not least, the output is controlled by the matrix, and switching output from ASCII to EBCDIC or any other code set requires a whole new matrix. Changing from QWERTY to AZERTY requires not only a change in the PCB layout (so that the tracks to Q and A and to W and Z are exchanged, to swap their matrix co-ordinates and thus their output codes), but the shifted numerals behaviour requires different logic for the shift key. The unhelpful "QWERTY" ordering of keys on a keyboard leads to an unnecessarily complicated matrix. Such keyboards also require a considerable amount of chips: 15 in the example in Cherry's 1971 brochure, and their PRO keyboard has been found with 14 and provision for 16 in total. Adding features such as serial output require circuit changes. Moving as much of the encoding and output processing into a single chip would go on to greatly simplify keyboard design and production.

RS 277-117

Radio Shack produced a TTL dual-mode ASCII keyboard kit for hobbyists. Part 277-117 comprised the PCB, one of the eighteen total ICs required (IC position Z1, RS74H103) and the documentation, with the remainder of the parts to be purchased separately. The documentation for this project provides good insight into how such keyboards were made, although it is quite a simplistic and limited implementation; available for download at Bryan’s Old Computers under Digital Group documentation, specifically the section *DG Systems*, the documentation includes the assembly instructions, complete theory of operation, output codes, circuit diagrams and troubleshooting. The kit was intended to be paired with part 275-1422, which appears to be a Hi-Tek High Profile array.

The age of the product is not known. The only known dated reference is in the Radio Shack 1977 catalogue, and the switch assemblies are likely to be surplus parts.

The output is split dual-mode. With Shift released, the alphanumeric keys output numbers and uppercase letters. With Shift held or Shift Lock engaged, the number keys produce symbols, but the letter keys produce control characters as though Control was held (the separate Control key has no influence on character output). The Shift keys themselves do not generate output of their own. Shift lock is electronic secretarial: press Shift Lock to engage, and either Shift key to release. The shift lock state is held in a flip-flop. The Control key does not affect the ASCII codes and must be detected specially; the same applies to the Break, Clear, Here Is keys and the two unmarked keys. These six keys produce bit patterns 1000 0000 to 1000 0101, with the eighth bit (called “E”) indicating that a special key was pressed; this is instead of giving the keys dedicated control lines on the edge connector. Two approaches are offered for dealing with these special keys.

The complete assembly required six capacitors, eleven resistors and eighteen ICs; the complete list of ICs is as follows:

Part number	Function	Quantity
RS74H103	Dual, edge-triggered J-K flip-flop	1
RS7473	Dual, J-K flip-flop	2
RS74193	Synchronous 4-bit up/down counter	2
RS7413	Dual, 4-input Schmitt trigger	1
RS74154	16-line demultiplexer	1
RS7475	4-bit bistable latch	2
RS7404	Hex inverter	2
RS7402	Quad, 2-input NOR gate	1
RS7400	Quad, 2-input NAND gate	2

RS7410	Triple, 3-input NAND gate	2
RS7420	Dual, 4-input NAND gate	2

Both positive and negative logic output is offered, on alternative lines on the edge connector (a complete set of lines for each of positive and negative logic, with the customer expected to determine which is required and use the correct set of lines). A Repeat key is provided to enable auto-repeat.

No explicit debouncing seems to be performed. The product documentation states:

> The keyboard scan principle, which has been used on all recent calculators, has the advantage of minimized effects of key bounce. (Key bounce is that tendency of a key to chatter, or double-entry, when pressed or when released.)

It’s interesting to note that this design requires more ICs than Cherry’s “PRO” keyboard, yet offers fewer features.

LSI-based matrix encoding

Various manufacturers produced off-the-shelf large-scale integration (LSI) matrix-based encoder integrated circuits. These are nearly complete keyboard encoder circuits contained in a single chip. Proprietary static encoders existed as early as 1969 at Micro Switch, while off-the-shelf scanning encoders were on the market as early as 1971, the earliest confirmed date for General Instrument’s AY-5-2376. Before long, numerous semiconductor manufacturers put out their own single-chip encoders, both static and scanning.

Certain aspects of the process such as controlling scan and debounce timing may need to be provided using additional components such as capacitors. These encoder chips often included a few thousand bits of ROM capacity for all the output codes for the keys; separate output codes would be defined for each of the supported modes. Alternatively, the chip could be designed to provide look-up offsets into an external ROM containing the character codes. The use of an external ROM (typically a PROM or EPROM) allowed the keyboard manufacturer to cheaply and conveniently customise the keyboard output without the expense of full mask programming of the encoder chip ROM.

Many characteristics of the encoding and output process are intrinsic to the encoder IC, and are impossible to change. These include the matrix dimensions, the rollover or lockout behaviour, the number of modes supported and the behaviour of the modes, the means by which modes are selected, and the number of bits stored per character. Many encoders connected directly to the matrix, with dedicated pins for each row and column. This reduced the overall cost by removing the need for external multiplexers or decoders, but disqualified those models if a higher key count was needed. National Semiconductor implemented a compromise with the MM57499: the physical wiring is for a 12×8 matrix (utilising 20 pins on the IC), allowing up to 96 keys, but by grounding the fifth strobe line, strobe lines 0–3 instead address a multiplexer to extend the matrix to 12×12 for a total of 144 keys.

The mode implementation also varied. National Semiconductor's MM5740 was a quad-mode encoder with 9-bit output, but bits 0–4 and 8 of each key were common to all modes, thus saving on ROM size and chip complexity and cost. Typically, though, each key could generate three or four fully-independent codes, one for each of the available modes. These codes could be set in the factory (typically to ASCII), or the customer could provide the manufacturer with the desired matrix map and have custom encoder chips fabricated to suit their equipment. As noted, it was not uncommon to provide the output tables in a separate chip and use an encoder with no internal tables.

The rollover and lockout behaviour was also encoder-specific. N-key rollover was widely implemented. A common alternative was N-key lockout: as soon as the first key in the matrix is detected, the scanning process stops until that key is released. National Semiconductor's MM5740 offered two-key rollover or N-key rollover, but this had to be set with the mask programming. General Instrument's AY-5-3600 and SMC's corresponding KR3600 as well as their KR9600 and KR9601 all permit a choice of rollover or lockout using the option pins. General Instrument's AY-3-4592 has a dedicated lockout/rollover pin. The literature for encoders indicates that, in N-key rollover operation, matrix scanning is suspended upon key detection, until the debounce time has completed for that key. That is, if multiple keys are pressed, they are not all debounced simultaneously, as that would introduce unnecessary complexity. Rather, the first key to be detected is debounced and output, then matrix scanning resumes. An area of RAM tracks the keys that were detected, so that they are not reported again on the next matrix pass. This way, keys are reported as they are struck, but holding them will not interfere with future keystrokes. Conductive keyboards require a diode at each position for this; electromagnetically-sensed keyboards (which some encoders such as General Instrument's AY-3-4592 support) do not.

Encoders were typically designed with parallel output, but certain models provided serial output instead.

See the keyboard encoder ICs page for details of some of the encoder models that were on the market.

A keyboard encoder would not necessarily provide the complete functionality of a keyboard, and additional functionality—beyond that provided by the encoder—had to be provided with additional logic circuitry. A good example of this is Jameco's JE610 kit: Jameco desired Repeat and Caps Lock functionality, and neither facility is included in the AY-5-2376 encoder that it uses. As a result, additional logic chips are required for these features.

Microcontroller-based matrix encoding

As the cost of integrated circuits fell, it finally became cost-effective to use a microcontroller as the keyboard encoder. On visual inspection, keyboards using a controller appear the same as those with an encoder (with both types having one or two large chips depending on whether the ROM is stored separately or integrated into the controller or encoder); some keyboards described as having a "controller" instead have an encoder logic chip, as described above. Sometimes the chip type can be determined readily from the part number (e.g. a model number from Intel's MCS-48 series or National Semiconductor COPS family) but other times the identity of the chip is not readily apparent.

Micro Switch claimed in October 1977, in an advertisment in Computer Design, to have introduced the world's first "intelligent keyboard", based on a microcontroller:

> Designing a full-function keyboard for your system used to have a major problem. A full-function price.
>
> MICRO SWITCH has changed all that with the intelligent keyboard. It's the first microcomputer-based keyboard. Ever.
>
> Which means when you equip it with virtually every function you can think of, it still comes in for the price of a simple encoded keyboard. Because all the functions that used to require extra components are now available on a single microcomputer chip.
>
> Besides traditional encoder functions, the chip can handle many others, such as parallel or serial data, multi-character storage and tri-stated outputs for direct data bus compatibility without using expensive I/O ports.
>
> The intelligent keyboard can perform more functions more efficiently because less hardware is needed. You get lower total system cost. Pin for pin compatible EPROM for faster design turnaround. Plus greater reliability since there are fewer components.

More details were provided in the December issue:

> Combining 103 Hall-effect logic scan key modules with a single-chip microcomputer results in a full-function keyboard for intelligent and distributed processing terminal applications. The microcomputer integrates 8-bit CPU, ROM, RAM, I/O lines, and a time/event counter on a single chip. A 40-pin EPROM, pin-compatible with the ROM, is available for prototyping. Features of the 19.38 x 16.34 x 1.44″ (49.23 x 16.1 x 3.66-cm) model 103SD24-1 keyboard include 4-mode, 8-bit ASCII code assignment; choice of serial or parallel data outputs; 14 relegendable keytops for programming keys; 8-deep FIFO character storage; N-key rollover; and timed/auto repeat for selected keys.

This technique, of writing the encoder as a software application for a microcontroller, was adopted wholesale across the industry and remains the means by which keyboards today are built. Compared to dedicated encoder chips, there is considerably more flexibility available: altering the behaviour of the keyboard is achieved simply by changing the firmware mask, and this permits signficant changes in logic to be achieved without having to redesign the entire encoder.

With their IKB-1 Intelligent Keyboard, also introduced in 1977 or 1978, IMSAI opted for an Intel 8035 microcontroller with a separate 4751 ROM chip for the program. The IKB-1 took advantage of having keyboard firmware to allow the keyboard's output to be reconfigured on the fly, primarily from the keyboard itself, but in certain cases also from the host computer. This included the ability to remap keys, although like all the other settings this was lost when the keyboard lost power.

C. P. Clare would also advertise intelligent keyboards in 1978, in both reed and capacitive forms. The type of microcontroller used is not presently known.

In the years that followed, a number of microcontroller families were used as keyboard controllers. Intel's MCS-48 family was a popular choice, with models 8039, 8048, 8049 being found in keyboards. Examples include Kaypro 2000 (8748), Apple Desktop Bus Keyboard (8048), Epson Q103A, Micronorth MCL-5165 and Ortek MCK-101 (8049), and Cherry KFN3-8358 (8039). The 8039 in KFN3-8358 is ROM-less, and is paired with an AMD 2716 EPROM for the firmware storage. Similarly, Cherry's UB88-0008 used a National Semiconductor COP402N ROM-less microcontroller paired with a National Semiconductor MM2758Q-A 8 kilobit EPROM. Fujitsu FKB4700 and Zenith 163-73 are examples of keyboards using an 8051 chip from Intel's MCS-51 family, the successor family to MCS-48. Sony's OA-S3400 word processor used a Fujitsu MB8841. The Commodore Amiga 1000 keyboard used a MOS Technology 6500/1-based controller; the 6500/1 is a microcontroller derivative of the better-known 6502 CPU.

Microcontroller-based keyboards have become the standard approach that is still used today. Microcontroller-based keyboards often communicate using a dedicated protocol. The "AT" keyboard protocol introduced with the IBM Personal Computer/AT (now the PS/2 keyboard protocol, with a smaller plug) introduced the ability for the computer to control the lock key status. Apple designed the Apple Desktop Bus (ADB), a universal daisy-chained peripheral bus used by their keyboards and mice, with automatic address negotiation provided by the firmware within each device. PS/2 and ADB were in turn superseded by USB. A single keyboard can communicate on multiple protocols. Some USB keyboards can be connected to a PS/2 port using an adapter, and they automatically detect what connection is in use. Some older keyboards had DIP switches to select between protocols such as XT, AT or PS/2, ADB and Sun. Having all the keyboard behaviour contained within firmware makes such designs far simpler and cheaper than logic-based encoding and dedicated communication circuitry.

Memory-mapped I/O

A rather intriguing take on this method was described in the article Microprocessor keyboard encoding in *Microprocessors* volume 2 number 2 from April 1978. The article was penned by Roger Munt of Terminal Display Systems Limited in Lancashire, England. Rather than use a microcontroller (even a ROM-less microcontroller), Munt's organisation decided to use a regular microprocessor, specifically a Signetics 2650. The 13 true address lines from the CPU (A0–A12) are fed into an 8T98 octal buffer, an HEF4515 4-to-16 binary decoder and the program and optional substitution PROMs. The eight data lines D0–7 connect in parallel to four hex inverting buffers (each HEF40098) and one or two PROMs. (Although the buffers are listed as HEF40098, which is a hex buffer, the schematic indicates that they are octal buffers.) The binary decoder selects where the data lines are connected: one of the four buffers, or one of the two PROMs. Thus, four of the address bits select which of the following is mapped into the address space:

- Matrix columns 1–8, via buffer 1
- Matrix columns 2–16, via buffer 2
- The control, shift and repeat keys, via buffer 3
- A DIP switch that selects odd or even parity, normal or TTY operation, serial or parallel operation, and baud rate, via buffer 4
- The 512-byte firmware PROM
- An optional 256-byte "substitution PROM" allowing single keys to output multiple bytes

Thus, the entire keyboard matrix is memory-mapped, using four address lines to enable the relevant buffer (for the desired half of the matrix) combined with eight address lines to isolate a specific matrix row. It does mean of course that the modifier keys and DIP options will span multiple addresses in memory. Debouncing is simply provided by waiting 10 milliseconds for the key to settle; no RAM or shift register is used for this purpose.

The use of a microprocessor means that a separate RAM chip is required, but in this case it appears that the firmware runs entirely from the CPU registers, with no dedicated RAM. An “option board” if fitted would contain RAM, to allow for more advanced functionality.

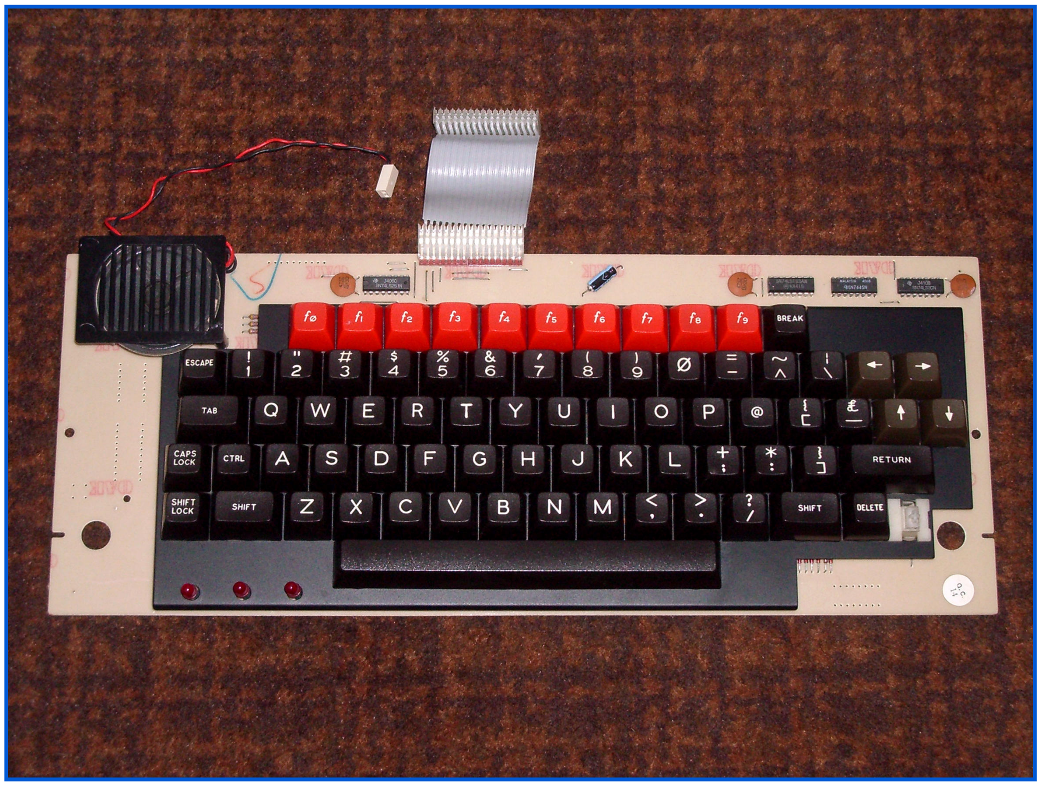
The article was written in 1978, and the keyboard was designed in late 1976, making it contemporary with the Micro Switch and IMSAI offerings described previously.

Host-based matrix encoding

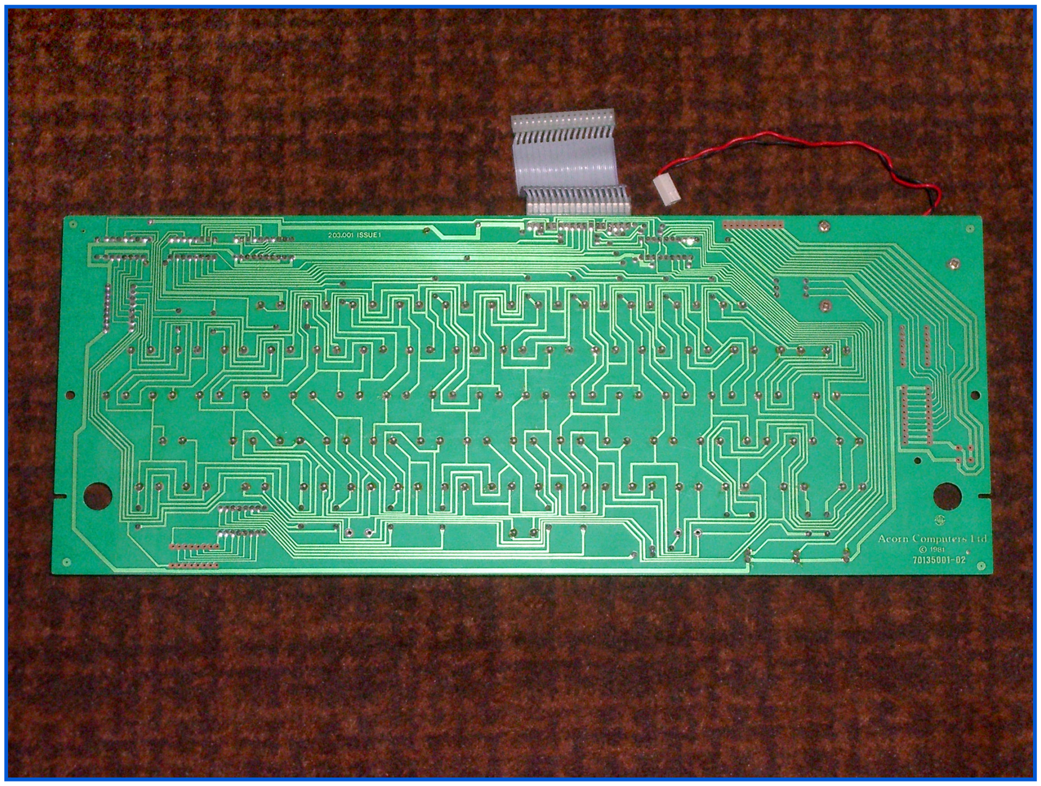
Host-based arrangements use the host equipment to identify the keys pressed. For example, the host could be a computer where the operating system is partially or wholly responsible for the keyboard operation. Where the keyboard is tied to the host hardware (such as a computer or terminal with an integrated keyboard), this offers a cost saving. The keyboard will need far more wires joining it to the host, but for an internal keyboard this does not inconvenience the user in the way that a thick and heavy external cable would do.

Acorn BBC Microcomputer

Acorn produced a hybrid software–hardware solution for the BBC Microcomputer in 1981. Because the keyboard is integrated into the case, placing a dedicated controller onto the keyboard would be wasteful. However, it would also be wasteful for the 2 MHz 6502 CPU to be continually used to check the keyboard. Additionally, there is a limit on the number of I/O lines available from the I/O controller to which the keyboard circuitry is connected. A compromise was reached where the complex processing is handled by the operating system, but the keyboard itself was responsible for detecting keyboard activity and multiplexing the matrix lines. A total of seventeen lines connect the keyboard to the motherboard, including the matrix row and column numbers (four and three bits respectively), the software-level reset line, the multiplexing clock, and the LEDs. The complete circuit diagram can be found on the BBC/Master Circuit Diagrams at MDFS, reproduced (uncredited) from page 490 of the Advanced User Guide (note that what appears to be “.5V” is really “+5V”). A simplified diagram is shown on the Arduino forum, but it mistakenly omits all the diodes and incorrectly shows all eight columns connected to the NAND gate.



Wong's–made ("type 1") BBC Micro keyboard with Futaba ML switches; the diodes can be seen at the bottom right, below the mounting plate



PCB of type 1 keyboard showing the matrix arrangement

The matrix itself comprises ten rows of eight columns. The rows and columns are both multiplexed by chips on the keyboard PCB (the row multiplexer is a BCD-to-decimal decoder). When no keys are pressed, the keyboard uses a binary counter to continually scan the matrix rows, fed by a 1 MHz clock signal from the motherboard. This counter controls the row multiplexer, and the columns are ignored. The matrix scanning is thus independent of the operating system.

Seven of the eight columns connect to an eight-input NAND gate. When a key is pressed, it brings that input to the NAND gate low, causing the NAND gate's output to go high: the gate output indicates whether any key in the row being scanned is pressed, without being aware of which key, as the column multiplexer is not used at this stage. The NAND gate output is used to signal an interrupt, allowing the operating system to become aware of keyboard activity. From this point onwards, the key-pressed interrupt is shut off, and the operating system scans the matrix directly by taking control of both multiplexer chips, giving it the ability to read each key position on demand.

Each key is assigned an internal scan code; these codes are held in a 70-byte look-up table in the operating system ROM, in 10-byte groups spaced apart at 16-byte intervals, interspersed with 6-byte blocks of data and code to make use of the spare bytes. This look-up table converts from the scan codes to ASCII codes. The scan codes themselves are binary-coded decimal (BCD), where the high nybble is the column number and the low nybble is the row number, taken directly from the values of the two multiplexer chips.

According to Professor Steve Furber, one of the engineers responsible for the design, the operating system handles de-bouncing by checking every millisecond to see that the keys remain held; the 1 ms timer interrupt is used by the operating system for this purpose. Once a fixed number of check cycles have elapsed, the key is deemed to be pressed, and if it remains pressed, auto-repeat is triggered. He also notes that once the last key is released, the operating system re-establishes the key-pressed interrupt and relinquishes the scanning duty back to the keyboard's on-board logic circuitry. Those who understand 6502 assembly language can determine this from the full MOS 1.2 disassembly at MDFS; do note however that some of the look-up table blocks mistakenly have the least-significant nybble of their address given as &A instead of &B.

Rollover is fixed at two keys, as two addresses in the 6502's zero page (used for the most frequently-accessed operating system data in Acorn MOS) contain the two keys currently detected, not counting control or shift. If the number of keys held simultaneously increases incrementally, the first two keys will be detected. Pressing a ghosting combination simultaneously may result in a ghost being one of the two reported keys, and games and other software that request specific matrix positions be checked will be subject to ghosting.

The keyboard design contains a couple of other curiosities. The final column of the matrix is isolated from the NAND gate, which instead receives a continual 5 V feed for its eighth input. Of this column, eight rows collectively form a single-byte configuration word in the manner of NVRAM (curiously, this is not done with a single eight-column row). The ability of the keyboard to scan any matrix position on demand allows it to read this configuration word during start-up. The eight bits can be set by wire links or by a DIP switch block, either option being fitted to two rows of eight holes on the PCB. This entire column is diode protected, as otherwise any start-up options set would result in ghosting. The remaining two row positions for this column are Control and Shift, also diode protected, and also isolated from the NAND gate. This indicates that neither Control nor Shift can alert the operating system; pressing for example Control+A would only generate an interrupt as A is struck, after which the operating system's controlled scan would detect that Control is held. These two keys will also not generate ghosts if they are held in conjuction with two other keys.

The operating system provides a call (OSBYTE 121) to allow user code to scan the matrix for a specific key or for any key above a particular scan code; a second call (OSBYTE 122) scans the matrix to locate any single key pressed above scan code 16, which ignores Shift, Control and the eight start-up options. Steve Furber mentioned that games are able to bypass the operating system and read the keyboard directly, which may improve performance by removing all the interrupts relating to keyboard handling. Note that one key is not present on the matrix at all, specifically the Break key which requests a restart of the operating system. The Break key is wired to its own reset line and functions independently of the rest of the keyboard. (Having a single key to restart the operating system proved troublesome and in the later BBC Master Series machines it could be locked out with a screwdriver.) Nonetheless, the user could still bind an action to that key, as it was a software-level restart rather than a hardware-level reset.

Radio Shack TRS-80 Model I

The earlier TRS-80 computer (retrospectively the Model I) used a simpler arrangement, as detailed in the TRS-80 Technical Manual (1978) on pages 23 (description) and 36–37 (schematic). The keyboard matrix is connected into lines A0–A7 of the CPU's address bus, and all eight lines (D0–D7) of the CPU's data bus, forming an 8×8 matrix. When line KYBD is signalled, the keyboard is made accessible via tri-state buffers. With all bits A0–A7 of the address bus set to 1, all matrix rows are activated simultaneously, and the data bus then indicates whether any key is pressed. If there are no keys pressed, the matrix does not need scanning, conserving CPU time. If a 1 appears on the data bus, then the process is repeated one row at a time in order to locate the key. The conversion from matrix position to ASCII code is performed by the computer's firmware. The KYBD line originates in a 74LS156 decoder/demultiplexer, driven from several lines of the CPU's address bus. Included in a brief list of problems with the computer on Wikipedia's TRS-80 entry is "keyboard bounce issues"; it's not stated whether the switch bounce time exceeded that of the firmware's debounce period, or whether debouncing was simply omitted by the keyboard scanning routine.

Identifying encoding types

Several encoding types are easy to identify: if there is a large array or grid of diodes, or photoelectric shutters, the method will be self-evident. Two-of-N keyboards with a discrete encoding grid are also easy to identify. Where there is only a single main chip, or an assortment of TTL components, the method is less obvious.

Keyboards with a microcontroller encoder most commonly use Intel MCS-48 and MCS-51 chips; look for part numbers containing models from those families, e.g. 8039, 8048, 8049, 8748 and 8051. The part number on the chip may contain additional prefixes and suffices that differ by second source manufacturer, e.g. Intel P8748AH, NEC 8048HC610 and D8048HC, and Signetics SCN8049H. There may even be an infix code, e.g. the Intel 80C51. Some practice is required to isolate the relevant digits.

A single chip encoder may also be a MOS LSI encoder; examine the list of known MOS LSI encoder models for a possible match.

In some cases it will be necessary to perform a web search for the model number on the chip to identify its purpose.

For TTL-based keyboards, drawing up an inventory of all the parts should give a fairly clear idea of the implementation method. The majority of the chips will be 7400 series TTL; the function of these chips can be located on the Wikipedia page list of 7400-series integrated circuits. Because 7400 series chips were made by a huge range of manufacturers, the part numbers take a number of forms; to identify a chip, ignore all prefix, infix and suffix codes. Motorola's MC74CH03 is simply a high-speed CMOS ("HC") variant of a 7403. The same applies to other product lines: a 9318DM is the ceramic-encased ("D") military-grade ("M") equivalent of the 9318PC plastic-encased ("P") consumer model ("C"): the model number itself is simply "9318", with manufacturers including National Semiconductor, Fairchild Semiconductor and AMD. Military-grade keyboards with 7400 series TTL will use 5400 series part numbers; these are functionally identical to the equivalent 7400 series parts. For example, SN5432J is a model 5432 quad 2-input OR gate and is equivalent to a 7432. On the Wikipedia list, the infix position is given as "x", e.g. for SN7404 (Texas Instruments–made hex inverter), discard the Texas SN prefix and add "x" after "74", giving "74x04". As with microcontrollers and encoders, some practice is needed to identify chip model numbers.

Matrix-scan keyboards will require a pair of binary counters to maintain and sequence the matrix co-ordinates. This may be two chips, or a single dual-counter chip. Matrix-scan keyboards also require a clock signal to increment the column counter and signal each successive switch scan; this task could be handled by a dedicated timer (e.g. a 555) or by a Schmitt trigger fed from an RC circuit, or it could receive an external clock signal. A crystal oscillator is also possible, but less likely. Matrix-scan keyboards typically involve a selector or multiplexer chip to select the row for scanning, and a decoder or demultiplexer chip to examine each column; regardless of the exact chip type used, there should be a method of multiplexing and a corresponding method of demultiplexing.

Two-of-N keyboards implemented in TTL will require three or four binary decoder chips, which is a significant clue in itself. Other clues include the use of double-pole switches whose two outputs are wired separately, and a lack of a clock generator.

Keyboards that fit inside a computer are quite likely to use the host to perform part or all of the scanning process. Some computers may delegate some simple TTL tasks to the keyboard circuit and handle the remainder in the OS or on the motherboard, while others simply connect the keyboard matrix directly to the motherboard.

Output

Data formats

Character codes

Keyboards of the 1960s and 1970s frequently reported the character codes of the keys pressed. These codes were often ASCII: pressing the A key would output 0x41 (ASCII 65, “A”) to the host equipment. Pressing the return would send 0x0D (ASCII 13, carriage return); alternatively you could send 0x0A with the Line Feed key.

The IBM Personal Computer XT—introduced in 1983—used a proprietary keyboard protocol based on scancodes; when the clone market took hold, the XT keyboard protocol became a de facto standard, as did the AT keyboard protocol after the IBM Personal Computer AT (released a year later in 1984) was cloned. Before this, ASCII was the de facto standard output. ASCII keyboards could be paired with the kit computers; the Apple I did not ship with a keyboard, and the customer could connect it to any ASCII keyboard available on the market. This could be a hobbyist kit keyboard (such as the RS 277-117 described above) or any off-the-shelf keyboard, including widely-available purpose-built and surplus units.

A major limitation of this approach is that keys not defined in the character set cannot be reported. ASCII has a limited repertoire of 32 special characters, including tab, carriage return and space, as well as special codes for transmission control. However, the functions available on modern keyboards, ranging from Print Screen to Japanese input mode, were all outside of the scope of ASCII.

Additionally, ASCII keyboards did not necessarily make the modifier keys known. Pressing the A key would generate 0x41 (“A”) while pressing Control+A would generate 0x01 (start of heading). Pressing 1 would generate 0x31 (“1”) while pressing Shift+1 would generate 0x21 (“!”). The keyboard did not give the computer a choice: the modifier key would be handled internally and the modified ASCII code reported. This is very similar to how many keyboards now have an Fn key that causes the scancode of the key to change completely, without the host system being aware. The accessibility feature known as “Sticky Keys”—where the modifiers can be pressed in advance by users unable to press more than one key concurrently—relies on the operating system being able to detect modifier keys separately, which is not possible with a standard ASCII keyboard.

ASCII keyboards also differed by whether or not lowercase was supported. It was normal for 70s computers to not understand lowercase: the 1976 Apple I, 1977 TRS-80 and Apple II and 1980 Acorn Atom for example were all uppercase only. The Shift key on uppercase-only keyboards did not affect the alpha keys. In Cherry’s *Switches & Switches Catalog C-73*, their product range included B80-3766 ASCII tri-mode with support for uppercase and lowercase letters, and B70-4753 ASCII quad-mode with support for uppercase letters only.

Keypress bitmap

A signficant disadvantage of character code output is that, regardless of rollover behaviour, the computer is unable to detect which keys are held. This greatly limits the ability to implement keyboard control of games. IMSAI introduced an alternative behaviour in their IKB-1 Intelligent Keyboard: the keyboard can be switched into “verbatim” or “unencoded” mode, either by the user or (in compatible configurations) the host. In verbatim mode, the keyboard continually transmits the status of the matrix, one row at a time. Each row is transmitted as a single byte, with four bits identifying the row (from 0–15) and four more bits as the bitmap for active keys in that row.

The use of a keypress bitmap is a permissible option for USB to achieve N-key rollover; the vast majority of USB keyboards use a bitmap only for the modifier keys, as the boot protocol for keyboards has a rollover limit greater than the keyboard itself.

Scancodes

Modern keyboards report key presses as scancodes. A scancode is an arbitrary but fixed code assigned to each key. Some keys such as Enter and F1 have scancodes with a one-to-one relationship to the function of the key. In contrast, the scancodes of alphanumeric and symbolic keys do not define the result of pressing the key. For example, AT scancode 0x10 will result in a *Q* being typed on a QWERTY keyboard, and an *A* on an AZERTY keyboard. The host computer can interpret the keys as it desires, by translating scancodes to any choice of output characters or actions. Localisation of the keyboard will still require separate key legends, but no electronics changes are necessary, as the operating system takes care of the final key encoding. A major advantage of scancodes is that they are capable of reporting non-printing keys, such as F1, Menu or Insert, that have no ASCII code.

Another advantage of scancodes is that they can simplify the logic circuitry on the keyboard. The keyboard’s on-board circuitry is no longer required to store or generate ASCII codes; it can simply report the values of the row and column multiplexers used to scan the matrix, combined into a single integer. The more difficult work of converting matrix positions to character codes could then be transferred to the host system; this can be seen in the BBC Micro example above, where the operating system uses a look-up table to convert matrix positions into ASCII codes. In practice, most scancode-based keyboards do still contain look-up circuitry from the matrix positions to the scancodes.

Scancode-based keyboards also report modifier keys to the host; this can be with scancodes (as in AT) or within a bitfield (as in USB). In AT for example, left shift is scancode 0x2A, while in USB’s boot protocol keyboard interface it sets one of the bits in the modifier bit field. This allows the operating system to detect modifier keys directly, and filter them out or allow actions to be bound to them (especially in games). This was often impossible in older keyboards, where the modifier keys were not reported at all.

Signalling

The simplest signalling option is simply to expose the PCB traces to the switches on an edge connector on the PCB. In this arrangement, there is no encoding at all: each switch has its own contact on the connector, with a common ground. Keypads often used this approach, although those too could be encoded. There are also unencoded matrix keyboards, that expose the matrix rows and columns on the edge connector, but these are typically proprietary units for use inside specific equipment.

The simplest option for encoded keyboards is very similar: an edge connector that provides all the bits of the character code or scancode. These can be positive or negative logic, as explained below. Such connectors may also have separate conductors for non-encoded keys, such as those without ASCII characters. Cherry’s B80-3766 ASCII keyboard has 7-bit output with parity, as well as a 12-key pad on the right of non-encoded keys.

ASCII keyboards could also provide serial output. In Amkey’s 1991 Electronic Engineers Master catalogue entry, they advertise TTL serial, RS-232, RS-232C and RS-422 models, with RS-423 also available. GRI also offered TTL serial, RS-232 and RS-422. Both manufacturers offered a choice of bit rates.

There are also dedicated protocols designed to handle keyboard operation. These include IBM’s proprietary protocols for the PC/XT and PC/AT, both widely adopted by clone manufacturers, and Apple Desktop Bus (ADB) used with the Apple IIGS and the Macintosh II and later Macintosh models. ADB was a general purpose 16-device desktop protocol with daisy chaining, while the PS/2 protocol introduced with the IBM computer of the same name still required separate mouse and keyboard ports, and peripherals would not function when connected to the wrong port.

Logic levels

Logic circuitry can take the form of “positive logic” or “negative logic”. In positive logic, *true* is indicated by a higher voltage (such as +5 V) and *false* is indicated by a lower voltage (typically 0 V). In negative logic, *true* is indicated by a lower voltage than *false*. Negative logic was created to accommodate older transistor technology, but as both methods have advantages, they can be used together. Hex inverter ICs are used to transform up to six data lines from positive to negative logic at once, and these are commonly found in old keyboards.

Keyboards that output scancodes or character codes directly need to generate the correct logic levels for the equipment to which they are interfaced: the output must use either positive or negative logic as called for. The type of logic used can be defined in the factory by the circuitry within the keyboard (and thus the user must buy the appropriate model for their needs) or, as with RS’s 277-117, both logic types can be provided simultaneously, with the user expected to wire up the appropriate outputs for their equipment.

Strobe

Keyboards with parallel output required some means to indicate to the host equipment when a key has been pressed. This was achieved with a strobe line from the keyboard: the strobe signal instructs the host to examine the data lines for a character code. The strobe line could remain active until the host acknowledged that the character code was read, or it could send a pulse only. This meaning of "strobe"—a data-ready indication—is different from the usage found in matrix scanning, of sequential addressing of a multiplexer. A common means of repeating a key was to simply transmit continuous strobe pulses, instructing the host to keep reading the same character off the data bus. In many cases this was done using a Repeat key that would drive a timer to repeat the strobe signal.

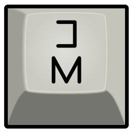
The host could expect positive-going or negative-going strobe pulses, and generic keyboards could be found with both strobe signals on separate lines on the connector. The strobe signal could be divided, with one side going through a NOT or NAND gate to invert the signal.

Keyboards designed to be integrated into a host device could instead generate an interrupt directly, such as with the BBC Microcomputer keyboard. AT and PS/2 keyboards also use CPU interrupts to signal keypresses, but these interrupts are issued by the I/O hardware inside the computer, rather than by the keyboard directly.

Modes

The historical term of "mode" is very similar to the modern term of "layer": it refers to the maximum number of different output codes available per key. The budget desktop keyboards supplied with computers and available in retail are normally single-mode, because the modifier keys are managed by the operating system. Laptop keyboards are frequently dual-mode, because they need an Fn key to simulate the keys that are not physically present due to size constraints. More expensive keyboards may also be dual-mode, to offer media control functionality on the Fn layer.

The mode count is much more significant in the days of TTL-driven ASCII keyboards. A single-mode keyboard would only generate one ASCII code per key: there would be no shifted characters of any kind. A dual-mode keyboard might offer a Shift key for extra symbols, or it may offer a Control key for generating ASCII control characters. A tri-mode keyboard could offer Shift and Control combinations but not both simultaneously: pressing Control+Shift+A would have to be interpreted as either Control+A or Shift+A (Cherry's B80-3766 gave precedence to Shift under these conditions). A quad-mode keyboard might extend the modifiers to permit Control and Shift simultaneously. For example, Cherry advertised B70 Teletypewriter Series quad-mode keyboards with uppercase-only output. The key marked "M"/"]" for example produces "M" when struck by itself, and "]" when used with Shift. Control+M emits 0x0D (carriage return), while Control+Shift+M equates to Ctrl+] and thus produces 0x1D accordingly. One keyboard with this functionality is their B70-4753 model.

	None	Shift	Ctrl	Shift+Ctrl	
	0x4D **M**	0x4D M	0x0D **^M**	0x0D ^M	— uppercase-only
	0x6D **m**	0x4D **M**	0x0D **^M**	0x0D ^M	— lower/uppercase
	0x4D **M**	0x5D **]**	0x0D **^M**	0x1D **^]**	

RS's 277-117 could charitably be described as 2½ modes: the Shift key produces shifted symbols or control codes depending on which key is pressed. This is instead of having separate Shift and Control keys, although there is a Control key. (The logic may have been designed for a different surplus keyboard which did not have such a key, or it may have simplified the TTL circuitry.)

The exact nature of the different modes is entirely implementation dependent. Micro Switch SW keyboard model 51SW12-1 is a non-ASCII keyboard described in the 1973 catalogue as dual mode: the first 6 bits identify the key location, and the 7th bit indicates Shift. At the same time, the SW brochure advertises it as 6-bit mono mode, presumably on account of the fact that the Shift key is signalled separately and does not influence the scancodes. Either view is correct, depending on whether you consider the scancodes to be 6-bit or 7-bit.

Bit-paired and typewriter-paired layouts

For many encoding techniques, it is considerably easier for the shift key to provide only bit manipulation. That is, the effect of holding shift and pressing a key results in a character whose bit pattern is derived from the bit pattern of the non-manipulated key. For example, self-encoding designs typically only permitted a single output code per key, and TTL-based encoders that used the matrix co-ordinates to form the output code had no means to hold additional codes for any key. As a consequence, the shifted outputs were defined by the effects of consistent bit manipulation on the ASCII codes of the keys. This arrangement is referred to as "bit-paired", as the uppercase and lowercase assignments are paired according to their ASCII codes, and hails from the Teletype ASR-33 with its electromechanical keyboard. The introduction of techniques such as ROM-based encoding allowed the uppercase symbols to be reverted back to how they were conventionally assigned on typewriters, giving "typewriter-paired" layouts: now the lowercase and uppercase codes could be stored in separate memory locations on the encoder ROM.

The table below shows the two shift assignments to the number row:

Number row key assignments

Typewriter-paired	!	@	#	$	%	^	&	*	(	)
Bit-paired	!	"	#	$	%	&	'	(	)	

Unshifted	1	2	3	4	5	6	7	8	9	0

The corresponding ASCII codes are as follows:

ASCII code comparison

Typewriter-paired	010 0001 (33)	100 0000 (64)	010 0011 (35)	010 0100 (36)	010 0101 (37)	101 1110 (94)	010 0110 (38)	010 1010 (42)	010 1000 (40)	010 1001 (41)
Bit-paired	010 0001 (33)	010 0010 (34)	010 0011 (35)	010 0100 (36)	010 0101 (37)	010 0110 (38)	010 0111 (39)	010 1000 (40)	010 1001 (41)	011 0000 (48)
Unshifted	011 0001 (49)	011 0010 (50)	011 0011 (51)	011 0100 (52)	011 0101 (53)	011 0110 (54)	011 0111 (55)	011 1000 (56)	011 1001 (57)	011 0000 (48)
Key	1	2	3	4	5	6	7	8	9	0

In bit-paired layout, shift+0 should be space, although further investigation would be required to see how different implementations coped with it (on the BBC Micro, shift+0 simply outputs 0). Also, the typewriter-paired example above shows the arrangement of a modern US keyboard; examination of old typewriters shows that shift+2 was more commonly the double quotation mark, and of course it was common for the 1 key to be omitted entirely from typewriters to reduce manufacturing costs. The UK keyboard layout retains the old shift+2 assignment, while typewriter-paired US keyboards map shift+2 to @ instead, allowing both single and double quotation marks to share a single key. ROM-based encoding makes such a change easy.

Key status

Typical USB keyboards (using the boot protocol) respond to the computer with the list of keys currently active (up to six normal keys and up to eight modifier keys). The AT protocol used on PCs before it (and still in limited use) sends a message to host each time a key is pressed or released. The host system (e.g. computer operating system) keeps track of which keys are active.

Some keyboards would instead only indicate keystrokes: the host would receive a message when a key was pressed, but not when it was released. ASCII keyboards operate this way. If auto-repeat is desired, this has to be provided by the keyboard itself. A common method for providing auto-repeat was to have a dedicated Repeat key which, when pressed, would retransmit the most-recently-pressed key. This is demonstrated in the RS 277-117 keyboard above; Micro Switch also offered this option.

Transmitting only the keys as they are struck (and not when they are released) was an approach used by Micro Switch with some of their keyboards, as a way to provide N-key rollover. This was an acceptable limitation for the time, but as computer prices plummeted and computers became widely available to home users, being able to detect which keys were held became necessary for any computers wishing to accommodate games with keyboard input.