µKenbak-1 Assembly & Operation Instructions

Up-to-date instructions are always available at www.adwaterandstir.com/kenbak

I would strongly suggest comparing the parts you received with the list on the next page. Let me know if you're missing anything and I will send a replacement.

PARTS LIST

Bag #1:
8 x 5mm White LED
4 x 5mm Yellow LED

Bag #2:
15 x tactile push button

Bag #3:
1 x 470Ω Resistor Array (10 pin)
2 x 2.2kΩ Resistor Arrays (10 pin)
3 x 10kΩ Resistor Arrays (6 pin)
1 x 16MHz Crystal
1 x MTS-102 Toggle Switch

Bag #4:
6 x 40mm F-F Hex Standoffs
6 x 12mm Pan Head Screws
6 x 8mm Pan Head Screws
4 x 14mm Flat Head Screws
4 x M3 Knurled Nuts
1 x 5 Pin Male Header
1 x 4 Pin Male Header (19mm)
6 x 4mm Nylon Spacers

Bag 5:
8 x black key caps
7 x white key caps

Unbagged:
3 x 16 Pin DIP Socket
1 x 28 Pin DIP Socket
1 x ATMega328p
2 x 74HC165 Shift Register
1 x 74HC595 Shift Register

1 x PC Board
1 x USB Extension
1 x USB to UART TTL Module
1 x DS3231 Module
1 x Aluminum Front Panel
1 x Project Case
2 x Aluminum Ends
4 x Rubber Feet

OTHER PARTS YOU MAY NEED

Soldering Iron with a nice fine tip
Good Solder (I recommend Alpha Fry Rosin Core 0.032”)
De-soldering Iron (optional)
Phillips Screwdriver
Needle-nose Pliers
Side Cutters (Nippers)

A word about soldering: Don’t underestimate the need for good solder and a good soldering iron. Most problems I’ve seen people have with this kit are caused by cold joints or insufficient wetting. That doesn’t necessarily mean you have to spend a lot of money. I’ve had good luck with $8 soldering kits from eBay. Just make sure it has an adjustable temperature and comes with an assortment of tips. Right now I’m using a $55 soldering station and it works great. I strongly advise you to get quality 60/40 Rosin core .032″ diameter solder (I use Alpha Fry). The spools I buy are only $10 and well worth it. I set my iron to 400C degrees and use the fine point tip.

Since the alignment of the LEDs and tactile switches is fairly important, we are going to put the tactile switches and LEDs in place (long leads facing to the right). The LEDs are in bag #1 and the switches are in bag #2.

Next, place the 4mm white nylon spacers in place (from bag #4).

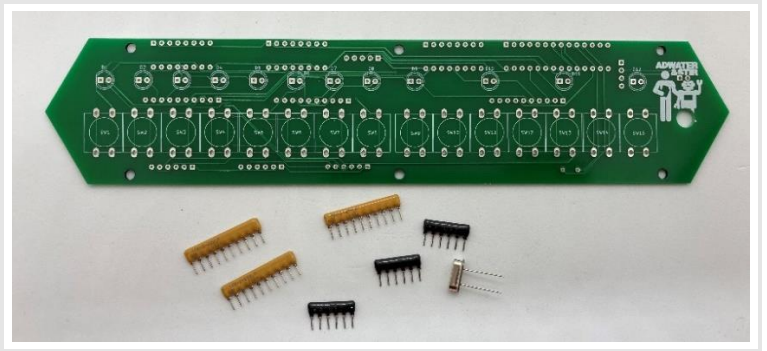
Put the front panel in place and secure it with six 12mm screws and the 40mm nylon standoffs (from bag #4).

Now you can rest the LEDs on your work surface and the front panel will keep them in place while you solder them.

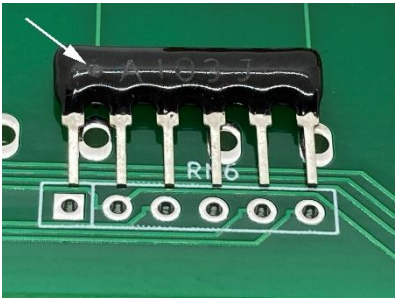
Check to see that the tactile switches are appropriately centered, and solder them in place.

After you have soldered all LEDs and tactile switches, remove the standoffs, screws, and front panel and set them aside for later.

Next we are going to install is the resistor arrays and crystal. These parts mount on the back of the printed circuit board (the side with the “Open Source Hardware” logo.)

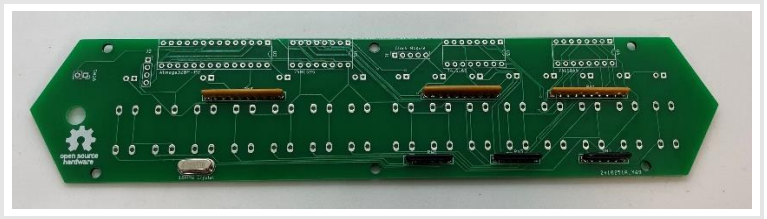


The 6-pin black resistor arrays are 10kΩ and the orientation is important. They are installed in RN2, RN3, and RN6. The end with the white dot should be installed in the square pad on the left side:

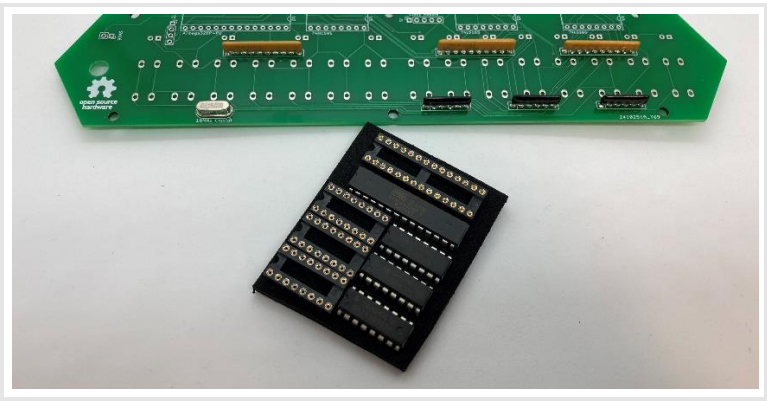


The two 2.2kΩ yellow resistor arrays are labeled “10X-2-222LF” and are installed in RN1 and RN4. The orientation is not important (they are isolated resistors, not bussed resistor arrays), but I like to install them with the dot to the left for consistency.

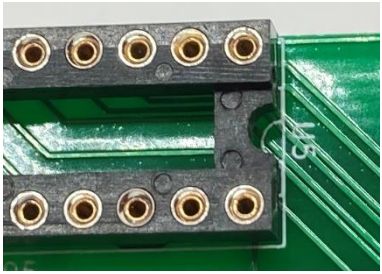
The yellow 470Ω resistor array is labeled “10X-2-471LF” and is installed in RN5. Like the 2.2k resistor array, the orientation is not important. Finally, add the crystal to location Y1. The orientation is not important.



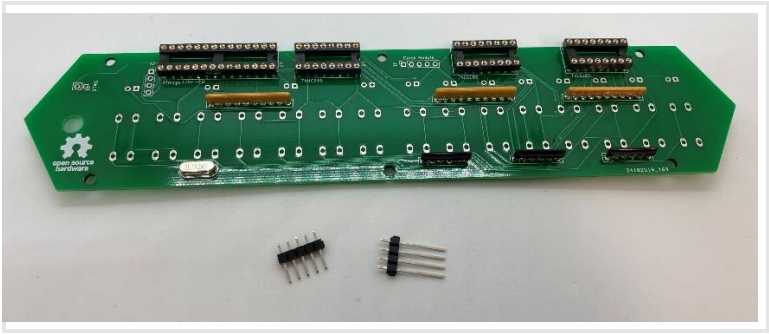
Add the DIP sockets to the back side of the printed circuit board (the side with the “Open Source Hardware” logo.)



Make sure the DIP sockets are mounted with the notch facing right as indicated by the screen printing on the circuit board.

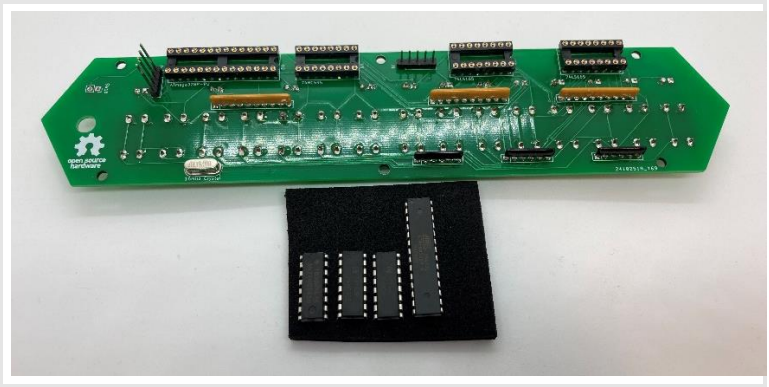


You’ll find two male headers, a standard size five-pin, and a longer four-pin.

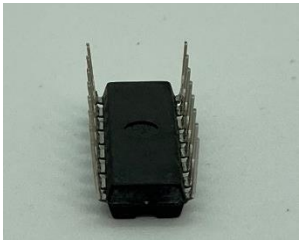


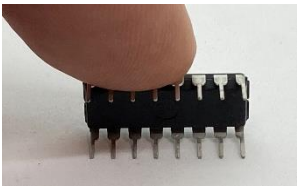
Add these to positions J1 and J2 on the back side of the board.

Install the integrated circuits.

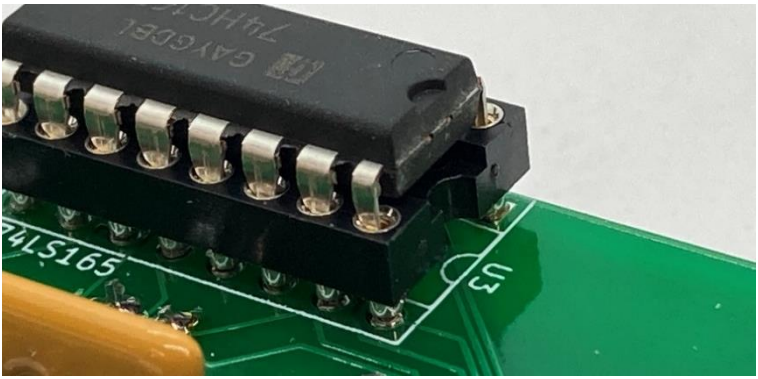


You'll notice the legs on the ICs are typically bent outwards just a bit. You will need to make the legs perpendicular to the body. There are IC insertion tools that do that, but I like to simply push down on a flat surface to align them.



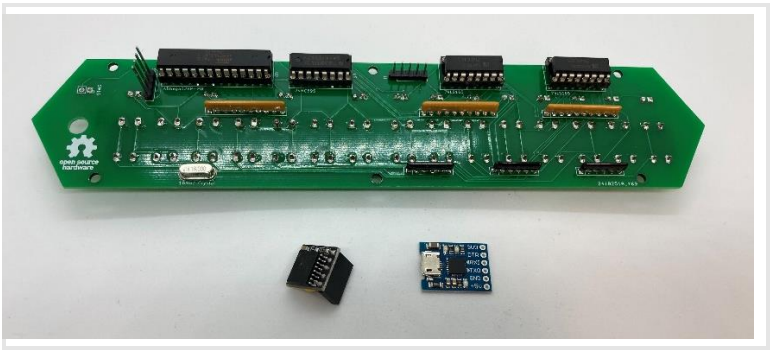


You will also want to make sure the notch at the top of each IC aligns with the DIP socket, and that the correct chip is placed where indicated on the silk screen.



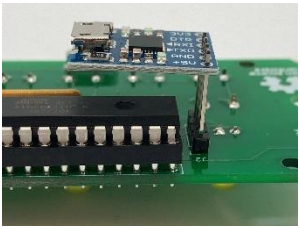
Note that there are two 74LS165, and one 74LS595. Make sure they are installed in the correct sockets as labeled on the circuit board.

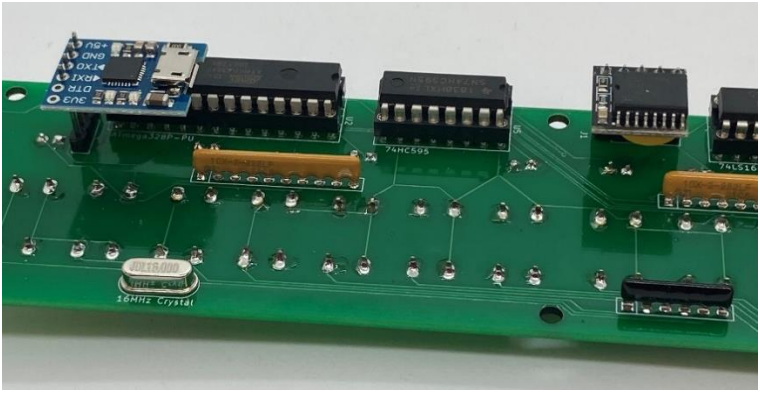
Install the DS3231 (real time clock) module and the USB to UART module.



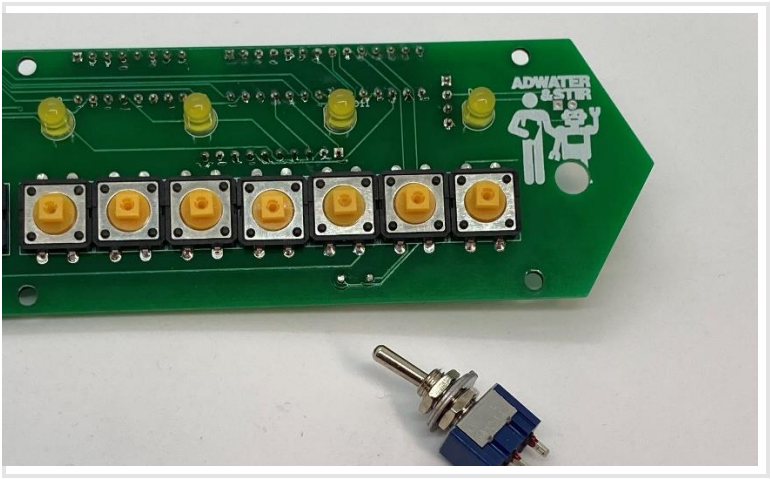
The DS3231 module simply mounts on the five-pin male header in the center, and the USB to UART module is soldered to the four-pin header (pins 5V, GND, RX, and TX as indicated in the photo.)

Leave clearance above the ATMEGA328p for the module.



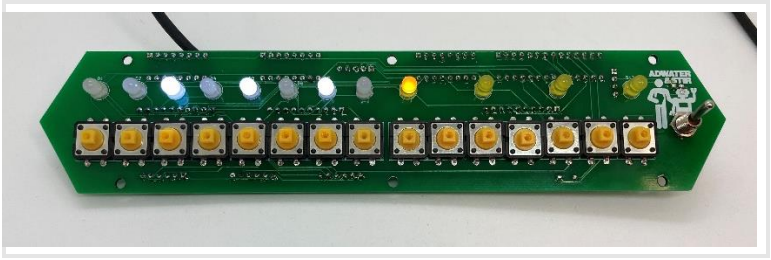


Add the MTS-102 toggle switch.



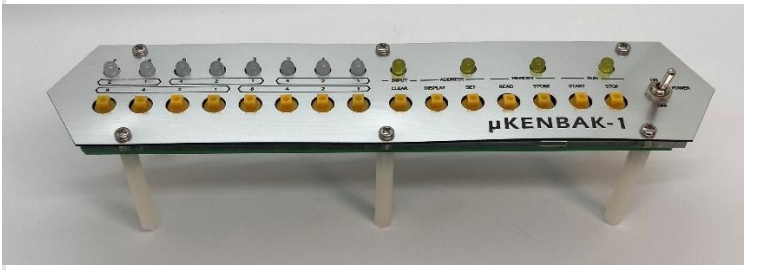
Remove the nuts and washers from the switch, secure it in place with a single nut, then turn the board over and solder a wire from SW16 to the center pin and bottom pin on the switch, as pictured.

This is a good time to check the function of your kit. You can plug a micro USB cable into the USB to UART module, flip the power switch up, and the µKenbak-1 should be fully functional.



Put the front panel in place and get the bags with hardware and the 40mm nylon standoffs.

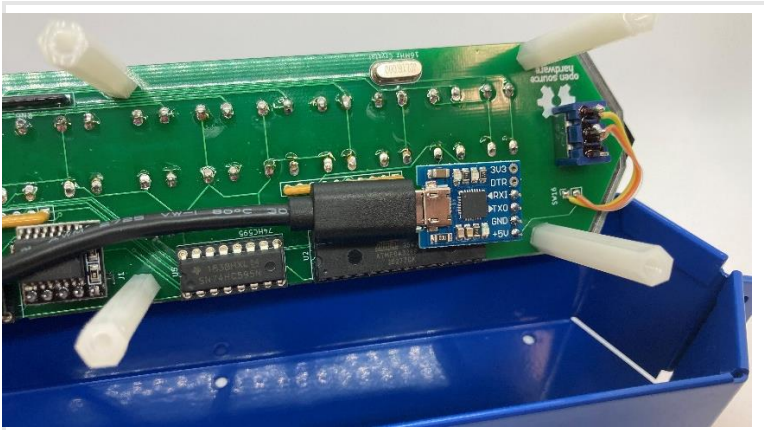
Add the 12mm pan head screws and standoffs as pictured:



Get the metal case and USB extension.

Secure the extension to the case with the supplied M3 bolts.

You can now connect the USB extension to the USB to UART module:



And secure the circuit board and front panel to the case with six 8mm bolts.

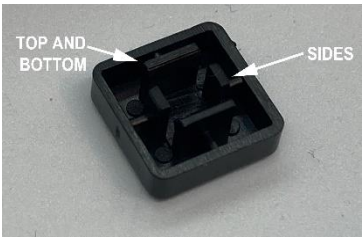
Next we'll add the aluminum side pieces.

The pieces are held in place by 14mm flat head bolts and brass knurled nuts.

If your side pieces seem loose, you can add a piece or two of masking tape under them to make them fit better.

Finally, we'll add the button caps.

The button caps will fit in any orientation, but I have found that it's best to have the buttons mounted as pictured. It makes it easier to pry the buttons off (from the top and bottom) if necessary.



CONGRATULATIONS! YOUR µKENBAK-1 IS COMPLETE!

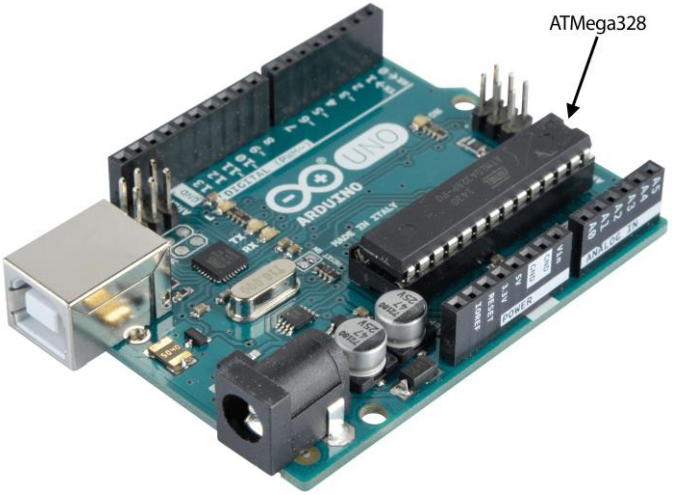
Read on to learn how to use a µKenbak-1.

Programming the ATMEL Microcontroller

Your ATMega328 microcontroller comes pre-programmed, but perhaps you'd like to dive into Mark Wilson's open-source code and do your own modifications.

First, you'll want to install the Arduino IDE (you probably already have this if you're actually thinking about modifying the code.) The source code is available at www.adwaterandstir.com/kenbak.

You're probably used to plugging into an Arduino and uploading the code, but there is no Arduino inside this project. There are other ways to upload the code using a programmer (I'll leave that up to you) but the easiest way is to simply remove the ATMega328 chip from the µKENBAK-1 and replace the ATMega328 chip in an Arduino Uno, then upload like usual.



Pop that chip out when done and put it back in the µKenbak-1.

History

This article was written by John Blankenbaker and was shared with me via email in July 2021:

John Blankenbaker was enrolled at Oregon State University from 1948 to 1952. During this period, announcements were made in the public media of new devices called computers. If there were photographs accompanying the story, they always showed massive pieces of equipment. There were no explanations as to how these machines actually worked.

In the summer between his junior and senior years John was enrolled as an intern at the National Bureau of Standards, as it was then called. There were approximately 100 student interns and they were assigned to positions randomly. John was one of four assigned to the SEAC project. SEAC stood for the Standards Eastern Automatic Computer. It was another large unique machine that had its own building to house it and the people who designed it and maintained it. SEAC was a working computer though modifications to improve its characteristics were being developed. Two of the take-aways were that SEAC was designed with Boolean algebra and that it was a stored program computer which kept the program of instructions (in the form of numbers) which told how to solve a problem in the memory with the data.

When John graduated from college one year later, he was employed by Hughes Aircraft Company. They had developed a computer for use on airplanes. This computer was not much larger than a breadbox, say a fat briefcase. Again it was a stored program computer which had been designed with Boolean algebra. Though the clock rate in the Hughes computer was slower than SEAC's, the machines were comparable in performance. At this time, vacuum tubes were the active power source and germanium diodes were the implementation of the

Boolean logic. These diodes were so expensive that each night an inventory had to be made of them.

Hughes Aircraft decided to design computers for use in the commercial market. This led to a design which was much more sophisticated than the airborne computers. It still used vacuum tubes and diodes but the input and output equipment was much more elaborate, including printers, card readers, and magnetic tape units.

Some of the commercial and military work was simpler than the large and expensive commercial machines. This led John to search for a simple computer. He did find that only one logic flip flop was needed. The description of a larger computer could be stored in the memory of a small machine. This small machine would evaluate what the larger computer, whose description was stored in its memory, would have found. This simple machine was described in the article “Logically Microprogrammed Computers.”

Smaller computers were available though generally beyond the price that an individual could afford. The alternative that was appearing was remote time sharing. Instead of having a computer, one could have a printing terminal connected by phone lines to a large computer where all of the logic and data was stored. These central computers would have programs such as BASIC. These arrangements were not inexpensive. The cost of the communications to the central computer could be significant if the connection time were extended.

John felt that the existence of a small computer, even with sharp limitations, could be an excellent training device for would-be programmers. Primarily intended for beginners, a small computer could also be entertaining for experienced people. John carried these ideas in his head for more than ten years.

Due to a management realignment at the company where he worked, John found that he was unemployed with a

small cash settlement. He decided that if he were ever to bring his ideas to market this might be the best time. In the last few months of 1970 and the first few months of 1971, he designed the Kenbak-1 computer which was publically shown in the spring of 1971. Over the summer, a corporation was formed and some capital was raised. With an ad in the September issue of Scientific American, the first unit was sold. Approximately forty machines were sold including three or four outside the USA.

The marketing emphasized schools which John later felt was a mistake. Budgeting for capital expenditures takes time and Kenbak Corporation was not that well capitalized.

- John Blankenbaker, July 2021

First advertised in the September 1971 issue of Scientific American (and later hailed as the "First Personal Computer" by the Boston Computer Museum), the Kenbak-1 remains a remarkable achievement!

Using the µKENBAK-1

Much of this text is from Mark Wilson's original documentation for his "Kenbak-uino" project from 2011:

Introduction

Not long after discovering the Arduino it seemed to me it could be a fun project to re-create an early computer, one with just LEDs and switches. I looked at things like the Altair 8800 (1975) but it has 30+ LEDs and 20+ switches and seemed like too much work.

Then I stumbled on the KENBAK-1 (1971). Perfect! Only a dozen LEDs and 17 switches. As a bonus it was the 40th anniversary of its introduction. I found a reasonable amount of

information online, starting at the Wikipedia article: http://en.wikipedia.org/wiki/Kenbak-1.

This is a software emulation of the KENBAK-1's CPU, and method of operation, together with a basic recreation of the hardware.

Operation

The basic operation is to enter values with the 8-bit switches on the left (0 through 7). CLEAR clears them all. The bits toggle on/off when a button is pressed. Pressing SET sets the address register to the displayed value. Pressing STORE writes the displayed value into the memory at the address and increments the address.

In this way programs can be entered. Like the original, the nanoKenbak-1 has only 256 bytes of memory and no registers, however some memory locations act like registers:

Register	*Address*
A	000
B	001
X	002
Program Counter	003
Output Register	200
Input Register	377

Also, memory locations 201, 202, and 203 were assigned to hold the overflow and carry bits from the A, B, and X registers, respectively.

START starts the program running.
STOP halts it.
STOP+RUN single-steps.
DISP displays the current address.
READ reads the memory at the address.

For more details - refer to the online information. For examples - see the scans of the original manuals available at

www.adwaterandstir.com/operation-kenbak. These manuals include:

- Theory of Operation Manual
- Programming Reference Manual
- Programming Worksheets
- Laboratory Exercises Manual

Extensions

Mark added a few extensions to the basic KENBAK-1 behavior.

Extension: SysInfo Instructions

The KENBAK-1 instruction set includes 3 NOOP (no-operation) op-codes: 02Q0, 03Q0 and 031R+[Second Byte] (Q=0..7, R=3..7). I've chosen a particular value of the latter, 0360, to implement an "operating system" SysInfo extension rather than no-operation. (I use 0300 when a real NOOP is required.) The execution of any of the NOOPs is handled by the virtual method

virtual bool OnNOOPExtension(byte Op);

on the CPU class. On the base class it does nothing (just returns true to indicate execution can continue). On the Sketch's derived ExtendedCPU class it traps the 0360 NOOP and execute the SysInfo function as follows:

The value in the A register sets the operation: if the high bit is set, the operation is a "write" otherwise it is a "read". The remaining 7 bits provide the Index of the item. The argument for a write comes from the B register.

The result of a read is placed in the B register.

Note that some writes actually perform an *action* and the corresponding reads do nothing. Executing the SysInfo instruction resets the CPU speed.

The first 8 values for the Index read/write the DS3231 RTC registers. Numbers are BCD (Binary Coded Decimal).

000: Seconds (00..59)
001: Minutes (00..59)
002: Hours (00..23) (always 24-hr, no matter how the RTC is configured)
003: Day (01..07)
004: Date (01..31)
005: Month (01..12)
006: Year (00..99)
007: Control
No validation is performed.

The next 8 values read/write bytes to the subsequent 8 bytes of "user" RAM in the DS2321:

010: Flags controlling the Kenbak-uino.
Currently only 1, if b0 is set, pressing one of the Data switches *toggles* the bit, otherwise it only sets it (as per the KENBAK-1).

011: EEPROM Page Map
See EEPROM Extension. The value of this byte defines how the 1k of EEPROM is partitioned into 8 pages starting with #0 of 256 bytes. Bits in the Map indicate if subsequent pages should be half the size, a 1 means halve, a 0 leaves the size as-is. The least significant bit applies to page #1 -- if it should be half the size of #0 (i.e. 128 bytes). Thus for example, a Map of 012 creates the following page sizes: 256, 256, 128, 128, 64, 64, 64

000 creates 4 full-size pages (higher pages are ignored): 256, 256, 256, 256

012: User #1
013: User #2
014: User #3
015: User #4
016: User #5
017: User #6
These 6 bytes are available for reading and writing non-volatile values.

The final 6 Index values act as follows:

020: Control LEDs
Reading does nothing. Writing sets the control LEDs as follows:
b0:INP
b1:ADDR
b2:MEM
b3:RUN
The upper 4 bits control the intensity of the RUN LED, 0000 = max brightness (PWM=255), 1111 = min (PWM=16).

021: Random
A read returns are random byte, 0..255 in B. Writing a 0 seeds the random number generator using the time etc. Writing a non-0 value uses that as the seed.

022: Program delay milliseconds
Reading does nothing. Writing delays execution for the given duration.

This is separate from the "CPU Speed" (Extension #7 below.)

023: Serial
Reads or writes a byte from the Serial port (@38400baud)

0177: Reading this special value simply returns 0 in Register A, indicating that extensions are enabled. Writing does nothing.

Extension: Blank

Pressing STOP+CLR (i.e. Press STOP and without releasing it press CLR) turns off all LEDs.

Extension: Erase

Pressing CLR+STOR sets all memory to 0 *except* 03 (P register) which is set to 04. The address register is set to 04, ready to enter a program. The CPU speed (BitN-STOP, Extension below) is set to 0.

Extension: Library

Pressing STOP+BitN loads one of eight pre-defined programs:

N	*Description*
0:	Simple counter
1:	Pattern
2:	Counting Clock
3:	BCD Clock
4:	Binary Clock
5:	Das Blinken Lights (random pattern)
6:	Sieve of Eratosthenes
7:	Set Clock

Extension: EEPROM

Pressing BitN+STOR writes program memory to EEPROM at "page N". Pressing BitN+READ reads program memory from EEPROM at page N. The ATmega328 has 1k of EEPROM, by default this is divided into 8 "pages" of various sizes:

N	*Size*
0:	256
1:	256
2:	128
3:	128
4:	64
5:	64
6:	64
7:	64

Program memory is always read from or written to starting at address 000.

Thus Bit7+STOR writes the *first* 64 bytes (addresses 000 through to 077) of program memory to EEPROM address 960. Bit0+READ reads all 256 bytes of program space (addresses 000 through to 0377) from EEPROM address 0. Note that EEPROM memory which has not been written to will be read as 0377 (Unconditional JUMP AND MARK INDIRECT).

Extension: SystemInfo

Pressing STOP+READ executes a SysInfo read. The Index is taken from the Address register; the result is placed in the Output register (0200).

Pressing STOP+STOR executes a SysInfo write. The Index is taken from the Address register (there is no need to set b7), the argument is taken from the Input register (0377).

All SysInfo calls are available programmatically and from the front panel but some make more sense that others (for example Delay from the front panel).

Extension: CPU Speed

Pressing BitN+STOP sets the "CPU speed". It sets the delay in milliseconds added after each CPU cycle, equal to 2^N ms. Thus b0+STOP sets the delay to 1ms. b7+STOP sets it to 128ms. The delay is set to 1 at power on and on CLR+STOR (Extension: Erase above) of if a program executes the SysInfo instruction, 0360.

Programs

--Count (STOP+Bit0):

Simply increments the OUTPUT register. Will be a blur unless slowed-down with BitN+STOP.

---Pattern (STOP+Bit1):

Cylon-style single LED moving left-right-left etc. Will be a blur unless slowed-down with BitN+STOP.

---Counting Clock (STOP+Bit2):

Alternates between showing the hours and the minutes. Minutes display in five minute increments (blinking LED = add five minutes.

---BCD Clock (STOP+Bit3):

Scrolls back and forth between showing the hours and the minutes in BCD. The minutes display has the left most (b7) LED blinking.

--Binary Clock (STOP+Bit4):

Shows the time in binary. The minutes are shown on the Data LEDs with the left most (b7) LED blinking. The hours are shown on the Control LEDs.

--Das Blinken Lights (STOP+Bit5):

Just blinks all the LEDs, old-school.

---Sieve (STOP+Bit6):

Builds a table of the odd numbers up to 255 and applies the Sieve of Eratosthenes to them. Then display the primes, HALTing after each one (hit START to proceed to the next one).

---Set Clock (STOP+Bit7):

Follow this procedure to set the time:

1. STOP+Bit7 to load the set clock program
2. CLEAR
3. SET (sets the address to 000)
4. Enter hours (24 hour format) in BCD
    - example: 0001 1000 (018 BCD = 6PM)
5. STORE (saves to A register)
6. Enter minutes in BCD
    - example: 0101 0100 (054 BCD = 54 minutes)
7. STORE (saves to B register)
8. RUN (runs the program, sets time to 18:54

Programming Example

Try entering this program into your µKenbak-1:

003 {Set}{Clear}
004 {Store}{Clear}

103 {Store}{Clear}
001 {Store}{Clear}
134 {Store}{Clear}
200 {Store}{Clear}
003 {Store}{Clear}
001 {Store}{Clear}
043 {Store}{Clear}
010 {Store}{Clear}
344 {Store}{Clear}
004 {Store}{Start}

The numbers are octal, and each octal digit is entered as a three-bit binary number. That's what the numbers under the LEDs are for. For example, the octal number 134 would be entered in the Kenbak-1 as

Yeah, programming was hard back then!

Serial Communications

The original Kenbak-1 did not have a serial port or any I/O method other than the incandescent lamps and push buttons. Because it was simple to implement, the µKenbak-1 does include a method to save and load data via the USB serial port.

Most computers will recognize the serial port when the µKenbak-1 in connected. If your computer doesn't, download

and install the CH-340 serial drivers. You can use a free utility like PuTTY, Tera Term, or CoolTerm to view/send the data.

Write out memory to USB serial port:

Press BitN+DISPLAY writes program memory as 16 lines of 16 bytes of octal data to the serial port. BitN sets the baud rate as shown below.

Read memory from USB serial port:

Press BitN+SET reads program memory from the serial port. BitN sets the baud rate as shown below.

By default the program expects octal constants, delimited by almost anything (comma, newline etc).

Prefix hexadecimal constants with 0x. Use uppercase A through F.

Each number is treated as a single byte and written to program memory, starting from address 000.

The operation halts when:

- the 256th byte is written to memory
- either an 'e' or an 's' (lowercase) is read from Serial (i.e. end/stop).
- STOP is pressed

When running, the program displays the most significant nibble of the current address (hex).

When finished it displays the length and checksum (sum modulo 256) in hex:

[0123456789ABCDEF] len=0x100 chk=0x9E

Examples

A file containing this:

```
0000,0000,0000,0004,0023,0220,0123,0000,0360,0023,
0221,0123,0000,0360,0023,0021,0360,0134,0002,0023,
0021,0360,0134,0001,0134,0000,0323,0017,0034,0001,
0023,0220,0360,0023,0222,0123,0024,0360,0024,0001,
0001,0034,0001,0023,0220,0360,0234,0200,0023,0222,
0123,0050,0360,0343,0016,s
```

or this:

```
0x00 0x00 0x00 0x04 0x13 0x90 0x53 0x00 0xF0 0x13 0x91
0x53 0x00 0xF0 0x13 0x11 0xF0 0x5C 0x02 0x13 0x11 0xF0
0x5C 0x01 0x5C 0x00 0xD3 0x0F 0x1C 0x01 0x13 0x90 0xF0
0x13 0x92 0x53 0x14 0xF0 0x14 0x01 0x01 0x1C 0x01 0x13
0x90 0xF0 0x9C 0x80 0x13 0x92 0x53 0x28 0xF0 0xE3 0x0E
e
```

will upload Das Blinken Lights.

Higher baud rates may be less reliable, although setting a transmit delay may help.

BitN baud rates:

- 0 = 4800 baud
- 1 = 9600 baud
- 2 = 19200 baud
- 3 = 38400 baud

Run Program at Power-On

You can set a stored program to launch at power-up.

To load a pre-defined program:

With the power off, press STOP+BitN, where N = the button representing the pre-loaded program (eg. STOP+3 for the BCD real-time clock.) While holding the buttons, turn the power on. Release buttons after the power-on "animation".

To load a user-saved program:

With the power off, press READ+BitN, where N = the "page" of memory you would like loaded. While holding the buttons, turn the power on. Release buttons after the power-on "animation".

To prevent any program from loading (default condition):

With the power off, press STOP. While holding the button, turn the power on. Release button after the power-on "animation".

To return all settings to defaults:

With the power off, press STOP+CLEAR. While holding the buttons, turn the power on. Release buttons after the power-on "animation".

K E N B A K u i n o