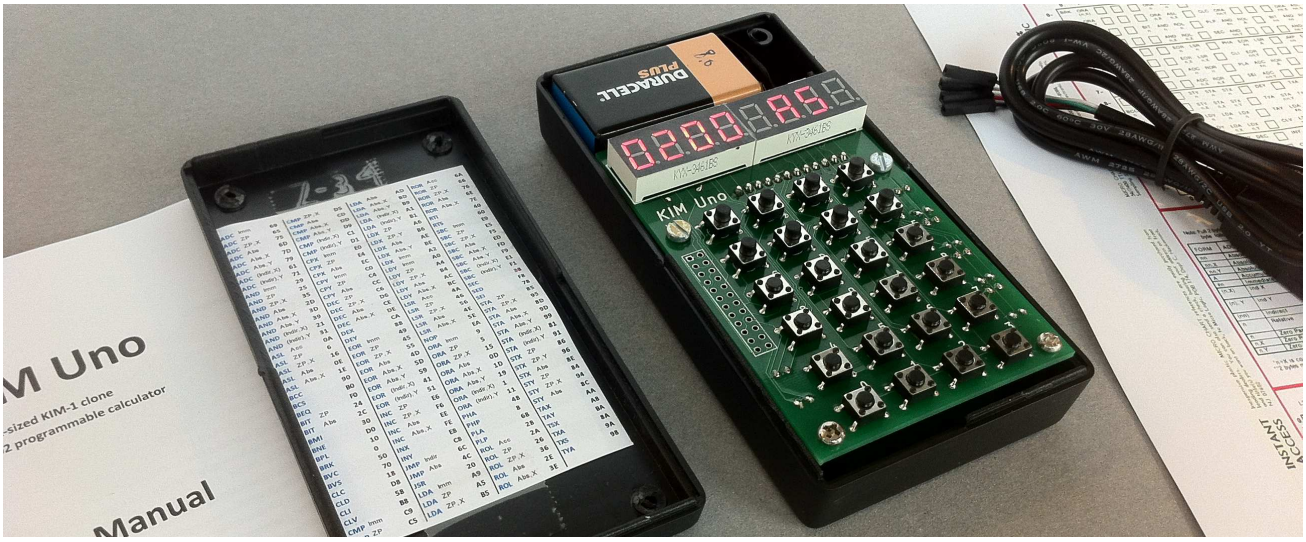
KIM Uno

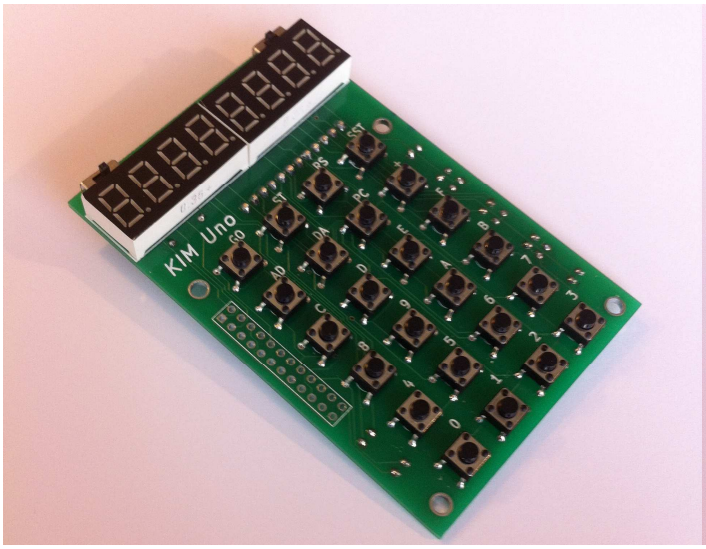
Pocket-sized KIM-1 clone &
6502 programmable calculator

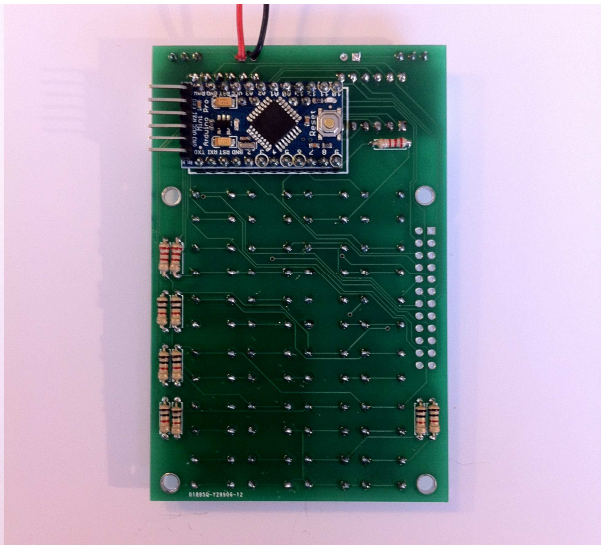
User Manual



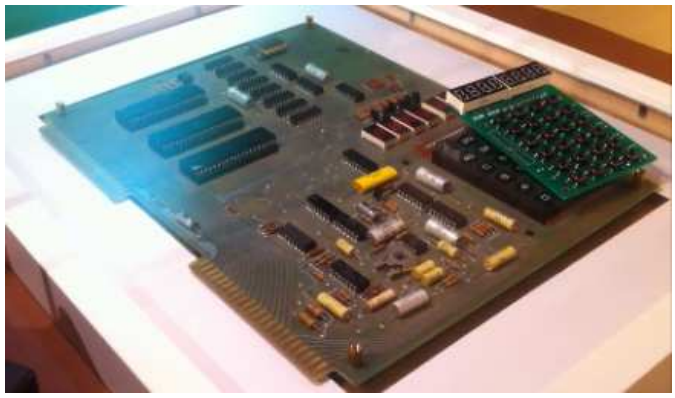
Obsolescence.wix.com/obsolescence

Contents





Introduction



The KIM Uno together with the original KIM-1

The KIM Uno is miniature replica of the KIM-1 computer. A 6502 programmable calculator mode is added as an extension of the original KIM. Despite its small size (similar to just the keyboard/display area of the original KIM), the Uno provides a faithful KIM 'experience'. An expansion port provides hardware hacking possibilities. The KIM's serial port is also present, so you can hook up the KIM Uno to a terminal/PC just like the original. Software archaeology is a major part of the fun. Some of the most interesting KIM software is stored in extra ROMs. So the KIM Uno plays chess, doubles up as a programmable calculator and contains some of the earliest KIM programming tools.

The KIM Uno is a very simple kit to build. The PCB contains 11 resistors, 24 buttons, a LED display and an Arduino Pro Mini. Building it requires no particular experience in soldering.

Why do this?

- Historical interest. The KIM was hugely important in early microcomputer development.
- Educational value. KIM coding makes you understand the inner guts of any computer.
- Appreciation of art. Enjoy the KIM’s beautifully minimalistic code like any other art form.
- Hacking fun. And my real KIM-1 no longer works.

History of the KIM-1

Imagine going back to 1976... A year ago, the Altair 8800 became the first commonly available microcomputer. A big $600 box, containing 256 *bytes* of RAM and no ROM. You toggle in your program bytes bit-by-bit. Alas, you can't save your program, unless you buy add-on boards that are just appearing on the market.
Enter the new KIM-1. Intended as a demonstration board by MOS Technology to showcase their brand-new 6502, it offers 1152 bytes of RAM. Plus 2K of ROM with a system monitor that is much easier than the Altair's front panel. And it has a serial port to connect a printer or terminal. And it has a cassette interface so you can save your programs. All that on a small board for $245. It was a surprise to MOS Technology, but with the benefit of hindsight it's obvious that the KIM-1 would find a huge market in hobbyist buyers. It started the second generation of microcomputers that was not only affordable, but actually usable without $1000 in add-on expansion boards. So the KIM-1 became the birthplace of many things, including the very first commercial games software (Microchess at $10) and the first software development tools. Commodore bought MOS Technology the same year, and built upon the KIM-1's momentum by introducing the PET and later the VIC-20 and C-64. Apple too built its empire on the 6502, as did Atari and Nintendo.

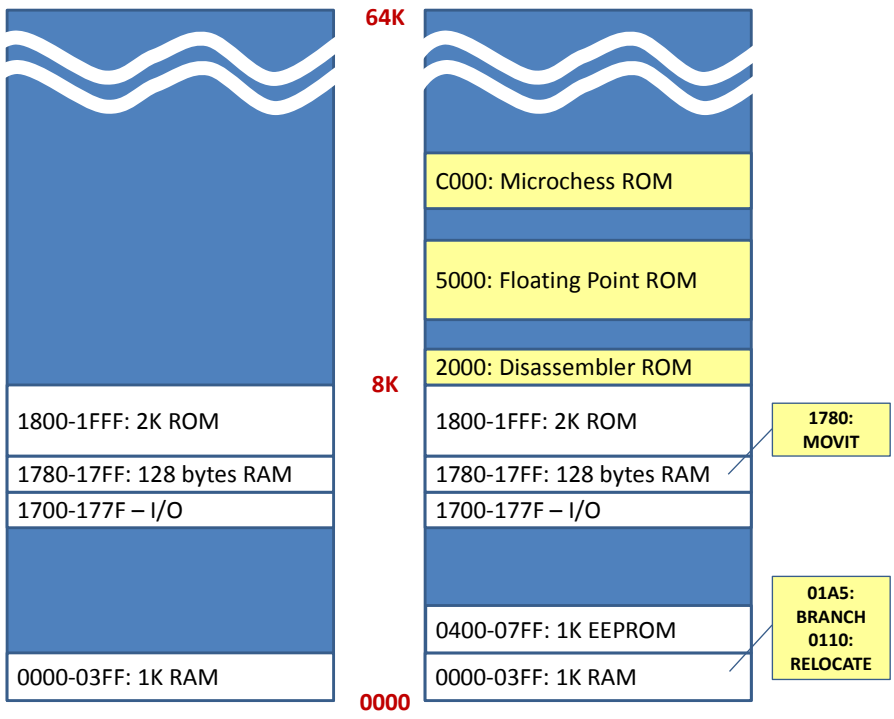
Summary of Features

- Open source hard/software.
- 6502 and KIM hardware emulated on an atMega328.
    - EEPROM serves as nonvolatile storage area *or* just as an extra 1K of RAM.
    - Serial port (ie, you can use a PC or terminal as external keyboard/screen).
    - Expansion port provides SPI and I2C. This makes it simple to add SD cards, WiFi or things like temperature sensors. Also, it allows any other Arduino to use the board when the on-board Pro Mini is removed.
    - Partial emulation of the 6530 I/O chips: the display is driven through ROM, but not through direct I/O register access.

- Extra software:
    - **6502 Programmable Calculator Mode**. A floating point ROM with 38 math functions is built in, and special display functions & instructions are added to make the programmable calculator easy to use from within the KIM-1.
    - **Microchess** built in to ROM. 'Dual screen' version with optional display of chess board on serial port.
    - Additional ROMs and preloading of software into RAM let the KIM Uno start up with a tool set that any KIM-1 programmer really needed back in the day: **Disassembler** (Baum & Wozniak), **Relocate** & **Branch calculator** (Butterfield), **Movit** (Edwards).

KIM Uno Memory Map

The figure shows the memory maps of the real KIM-1 and the KIM Uno next to each other. The core point being, ROMs have been added in unused parts of the KIM's memory map. And some smaller utilities are preloaded into RAM at boot time; these can be overwritten if not used. Back in the day, they were typically loaded from tape before starting to code on the KIM. Having them preloaded is a much nicer option.



Technical Details

The atMega328 microcontroller used in the Uno is close to the minimal spec of running a KIM-I emulator. When Mike Chambers published a 6502 emulator for the Arduino, the KIM Uno evolved from that base. Some background is given in the blog posts mentioned in Appendix 1. Below is an outline of the end result in terms of hard and software. All source code as well as the schematics and Gerber files are available as downloads on the Obsolescence Guaranteed web site. **Proper credit where it is due:** The bulk of the source code consists of Mike Chambers' 6502 emulator, and of course the original KIM-1 ROM.

1. Software

The KIM-1 basically consists of a 6502, two 6530 RIOT chips plus 1K of RAM. Each RIOT contains 1K of ROM, 64 bytes of RAM and I/O & Timer ports. The KIM-1 ROM code was added into the 6502 emulator's memory map, two 64-byte RAM spaces were added and of course 1K of Arduino RAM is used as main memory. The Arduino's built-in EEPROM is used as the second kilobyte of memory, and a write-protect can be switched on or off.

Having done that, the functionality of the I/O and timer ports on the RIOts needed to be replicated. The current version of Kim Uno is a bit lazy. The 6530 timers respond with TimeOut. And the SAD IO port responds with 0x01 to indicate that the KIM's ROM should use the onboard keys/LEDs, or (after pressing [TAB]) with 0x00 to indicate the user wants to use his serial terminal. That lazyness has very little impact on functionality, because the emulator intercepts calls to the screen and keyboard routines in the KIM ROM, and does its own screen/keyboard I/O in a way invisible to the ROM.

KIM Uno with a 1976 vintage ADM 3A terminal - using a TTL to RS232 converter

Almost 40 years after the original KIM-I, there must be something improved in the KIM Uno? At boot time, demo programs are loaded into 0x200 and 0x210 for immediate gratification purposes. Also, the NMI and RST vectors at 0x17XX are set during startup. You used to have to do that manually every time to make BRK and SST work.

The famous programming utilities ***disassembler, relocator, movit*** and ***branch*** are added, either as extra ROMs or as preloaded code in the KIM's RAM. A floating-point library, **fltpt65**, is also added in ROM, allowing the KIM to become a 6502 programmable calculator. The emulator supports this with a pop-out-of-KIM-emulation "Calculator Mode" in which the KIM's display is briefly extended with an extra digit and decimal points to view floating point numbers comfortably. And of course, **microchess** is built in to ROM.

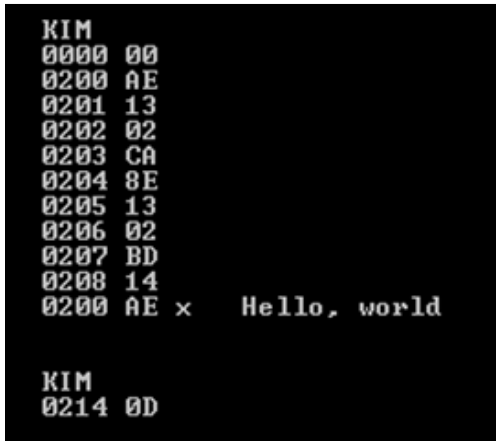
Running the emulator on any Arduino (with or without the KIM Uno board)

Actually, the software will run on *any* Arduino. Without a physical keyboard and LED display, it will only let you use it over the serial port of course. You can choose between two 'Modes'. Mode 1: use a terminal program such as puTTY (9600bps) to look at the six digits like they would be present on the physical hardware, or (Mode 2:) just press [TAB] to switch the KIM-1 to its normal serial mode operation. In Mode 2, the KIM-1 ROM expects slightly different keystrokes from the user. See the KIM-1 User Manual for details.

KIM Uno over the serial port: using Mode 1 (emulation of onboard 6 digit LED display)

Note that in mode 1, the KIM-1 ROM *thinks* you're using its on-board keyboard/display, not a serial port terminal. It's the emulator that reroutes the output to the serial port. The following ASCII terminal keystrokes simulate the buttons on the KIM-1 keyboard:

AD - Ctrl A	**ST - Ctrl T**	**SST on -]**
DA - Ctrl D	**RS - Ctrl R**	**SST off- [**
PC - Ctrl P	**GO - Ctrl G**	



KIM Uno over the serial port: using Mode 2 (using the KIM's serial teleprinter functions)

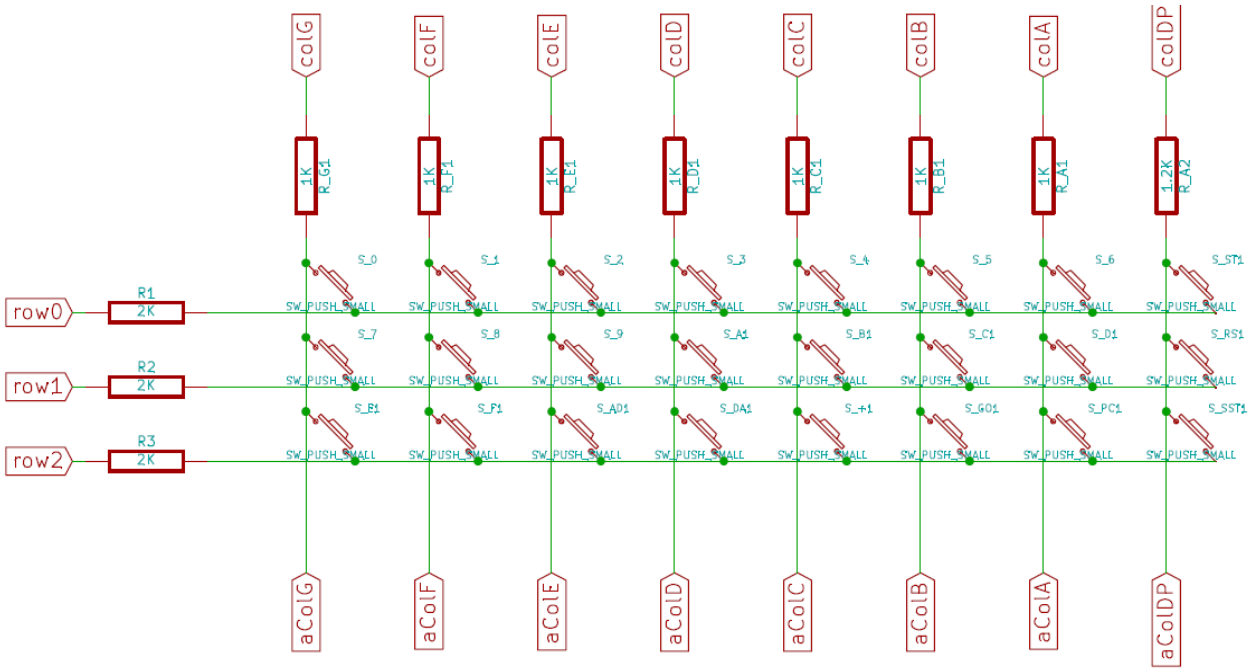
In mode 2, you are using the KIM's luxury mode. Back in 1976, it was not likely, but possibly you were rich enough to afford a teletype printer or even a serial terminal. Comfortable ASCII keystrokes were then used instead of the on-board dedicated keys. And of course the KIM-1 could print text to the terminal, instead of just flashing the contents of 3 bytes on its LEDs. Hit [TAB] at any time to go back to Mode 1.

2. Hardware

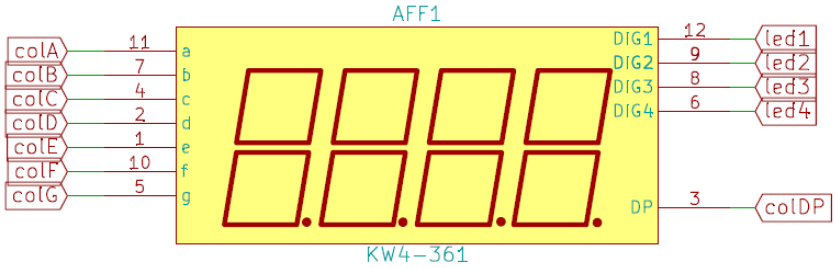
The KIM-1 has an ingenious schematic to use mostly the same I/O pins to both scan its keyboard, and drive its 6-digit LED display, flipping the I/O pins from input to output mode. There are three groups of I/O pins:

- 7 '**column pins**' which sense 7 keys on the keyboard, and then quickly flip over to drive the 7 LED segments that together form a LED digit.
- 4 '**row pins**', output pins that provide power to a row of 7 keys each. The KIM cycles through each of these rows to scan for a keypress (i.e., a short between that row line and one of the column pins).
- 6 '**led pins**' that provide power to one of the six LED digits on the display. They are the anode (+ side) for all the seven segment LEDs that together form one LED digit. The column pins are the cathodes.

Flash the 6 digits in sequence fast enough, and the human eye will see all six LED digits light up at the same time. Scan the keyboard in-between lighting up each LED, and the keyboard/display comes to life! The original schematic for doing all this is beautiful in its simplicity, and with the atMega's extra driving power it can be simplified even more (by leaving out buffer ICs and driving transistors) to *exactly* fit the Arduino's I/O space. Fitting in with the minimalist approach, only 11 resistors and 2 four-digit segment LED blocks are needed. See Appendix 2 for the full schematic. Shown below are the keyboard and LED parts.



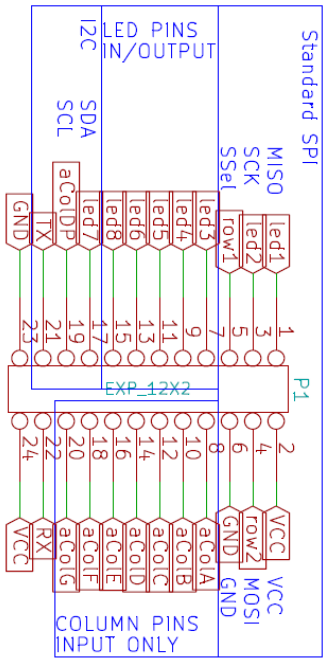
Above: the keyboard matrix schematic. At the bottom are the column pins, which sense keypresses. To the left are the row pins, which are powered consecutively to power a row of keys, looking for keypresses. Below is one of the two LED display blocks. On the right, four led pins provide power to each of the four led digits. They are switched on consecutively by the KIM ROM. On the left, the column pins also used in the keyboard circuit are now in output mode. If they are set Low (0V), one segment of a LED digit lights up. If they are set High (5V) no current flows through from the led pin.



You may recognise two changes from the original KIM-1 circuit... the following Arduino pins are used:

- **D2..D8**: **Column** pins used to sense keyboard and to select the LED segments.
  A5: An **extra** column pin not found on the original. An extra column is needed to drive the decimal point - which is the 8th segment of a LED digit. At the same time it was needed to hook up the special keys ST, RS and SST. On the KIM-1, these may look like keys but they are actually signals directly to CPU pins (RST, NMI). On the KIM Uno, these are all on Column 8, of which the KIM-1 emulator of course knows nothing - and needs to know nothing.
- **D9..11**: keyboard **rows** 0,1 and 2
  Identical to the KIM-1 schematic. The KIM-1 had a fourth row, used only to sense the serial port on/off jumper. That jumper wire had to be soldered on to enable serial terminal I/O. It's replaced by a keystroke [TAB], handled in software. So row 4 became a virtual IO line in the KIM Uno.
- **D12, D13, A0..3**: **LED** digit select lines for the 6 KIM-1 digits
  Optionally, **A4**: LED digit select line for an additional 7th LED digit used in calculator mode.

Expansion Connector:
Adding I2C & SPI to the KIM Uno as an option

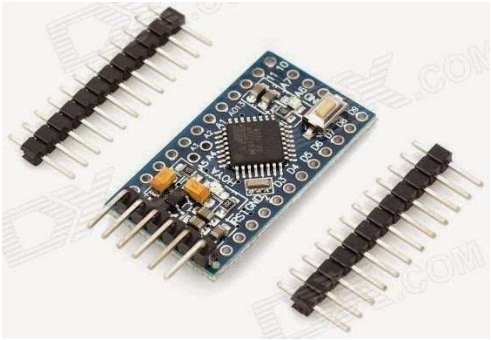


The above leaves A4 free unless 7-digit calculator mode is used. And A5 is only used for output to a sense line consisting of a row of keys. Nothing gets hurt if another output purpose is fulfilled by A5 when not scanning that row of keys. This is nice, because A4 is the I2C Data line (input/output) and A5 is the I2C clock line (output only). So the KIM Uno can have an I2C expansion interface adding more fun in the future.

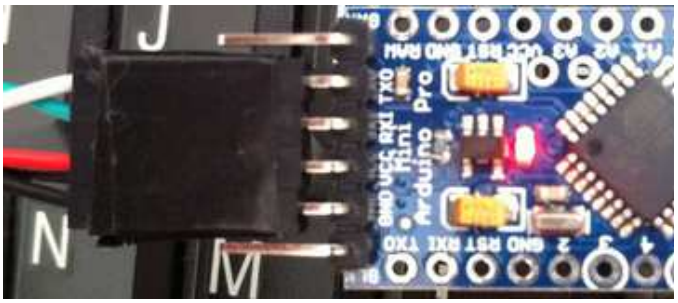
Using SPI is also a possibility. But it will give some of the I/O pins double duty. I.e., you have to stop scanning the keyboard and driving the LEDs to use the SPI port. And peripherals on the SPI port will get (unproblematic) meaningless signals on their lines if the LED or row pins are used for the display/keyboard.

The figure to the left shows the pinout of the expansion connector. The top 3 rows are a standard 6-pin SPI (pins 1-6), I2C is on pins 17+19. The other pins can be used for extra SPI chip selects or whatever, at times when the KIM Uno keyboard/LED display is not used.

Choice for the Arduino Pro Mini



Part of the fun in this project was to minimise building costs. A complete Arduino Pro Mini costs less than a single, separate atMega328P DIP chip. The Mini Pro has the footprint of a normal 24-pin chip but includes crystal, capacitors and the serial port connector in one go. By putting it on the bottom side of the PCB, underneath the segment LEDs, a lot of PCB board space can be saved, minimising the size and thus cost of the PCB.



Connector Pinout: The easiest cable is a TTL-to-USB cable, which provides the serial connection as well as a 5V power source. As these cables often lack documentation, here's a picture of how to match the colours to the Pro Mini's pins. Black is GND, red is +5V, green is TX, connected to the Arduino's RX, white is RX, connected to the Arduino's TX.

Downloads

Current version of firmware, schematics, gerber PCB designs and the 6502 software can all be downloaded from the Obsolescence Guaranteed web site. See Appendix 1 for details.

How to use the KIM Uno

The KIM's monitor program is actually pretty user-friendly. That is, once you understand the basic ideas of Address and Data modes and the purpose of the PC button....

To start off the Quick Introduction below, shown below are the KIM-1 and KIM Uno keyboards. Following after that Quick Introduction, this page also describes some of the extra features in the KIM Uno, such as using the EEPROM memory space. Also, three important utility programs from the First Book of KIM are built in to the KIM Uno. Movit, Relocate and Branch are explained at the end of the page.

1. Quick Introduction to the KIM-1 ROM

To start, enter [AD] 0200

- Hitting [AD] puts you in Address Mode. In other words, you type 4 digits to select any address in the KIM's memory.
- You now see that address 0x0200, which is where you normally start your user programs, holds the value A5. Search for $A5 on the chart in the Appendix, or on www.obelisk.demon.co.uk/6502/instructions.html, which is a great site to learn 6502 coding. A5 is the code for LDA...

Press [+]

- The Plus key just increases the memory address by one, so you can view the next byte. This way you can step through memory to look at it.

Now enter [DA] A5

- [DA] puts you in Data Mode. Instead of your digits ending up to form an address, they now form an 8-bit value that can be stored in the memory address shown in the leftmost 4 hex digits. As you typed in A5, that'll overwrite the old value.
- Hit [+] to go to the next memory address, enter two digits, save them with [+], two digits, [+]... this way you can quickly enter a program that you may have assembled on your PC or whatever. By the way, a good interactive, online assembler to quickly cook up code is www.masswerk.at/6502/assembler.html. Just write the assembler code, hit the Assemble button on the middle of the page, and start entering the byte codes at the bottom of the browser page into your KIM.

Switch the Kim Uno off & on.

- This is just for restoring the demo program that KIM Uno boots up with in 0x0200-0x0209, it might have been damaged by your tinkering just now. This is actually the very first program from the First Book of KIM. It swaps the value in 0x0010, which happens to be 0x10, with the value in 0x0011, which happens to be 0x11.

Enter [AD] 0010

- Observe that address 0010 holds the value 10. Hit +. Notice 0011 holds the value 11. Hit 0200 to go back to 0200 again. You'll see 'A5' again.

Press [GO]

- The program runs and the address shown afterwards is 002A, which is the first byte after the program's end.

Press [AD] 0010

- 0010 now holds 11 instead of 10, and if you hit [+], you'll see 0011 holds 10 instead of 11. Indeed: the values have been swapped by running the program.

Press the [SST] button (original KIM-1: slide the SST switch to the On position)

- Type 0200 to go back there, you're still in Address mode.
- Press [GO] and notice that only the first instruction is executed. The display now shows 0202 A6, which is the address and code of the following, second instruction (LDX).
- But to further inspect the CPU state after the initial LDA instruction that you just executed, you can also explore the following memory locations. They hold a copy of the CPU registers after running that first instruction just now.So press [AD] 00F3 and see that the accumulator holds 0x10, etc. You can also edit these stored register values using the [DA] key.

```
00EF = Program counter Low byte   00F0 = Program counter High byte
00F1 = Status Register (P)        00F2 = Stack Pointer (SP)
00F3 = Accumulator (A)
00F4 = Y Index Register           00F5 = X Index Register
```

Press [PC]. This will show you the program counter is still at 0202, ready to execute the second instruction. Hit [GO] to execute it. Keep hitting [GO] until you are at program end (0208). Get out of SST mode by pressing [SST] (or set the slide switch to OFF on the original KIM-1).

Above, you've gone through the entire programming interface of the KIM. Congratulations, all you need to do now is memorise the 6502 instruction set in byte code, using www.obelisk.demon.co.uk/6502/reference.html to search for mnemonics and byte codes. Or if brain capacity is scarce, use www.masswerk.at/6502/assembler.html as an assembler and type in the byte codes it generates for you.

You might still wonder about a few things:

- you can stop your program using the [ST] key, where ST stands for STop your program and in the ROM source code, it stands for STart the monitor. Same thing.
- the [RS] key is a normal Reset button. Note that it will not erase memory, and in fact leave many other settings like they were before. But you enter the KIM Monitor this way if the [ST] key is not strong enough. By the way - [ST] triggers a NMI, and [RS] a RST signal to the 6502.
- how to set break points like a debugger provides? Simple: enter value 00 (for the 6502's BRK instruction) in the address you want to generate a break point at. Do not forget to note down the original value in that byte, and put it back afterwards. Value 00 is the BRK instruction, and it drops you in the KIM Monitor just like Single-Step mode does. So inspect the CPU registers, change them as you like, and hit [GO] to continue. But remember you nuked an instruction by overwriting it with a BRK instruction. Real KIM Progammers used to enter spare NOP instructions at points where they thought they might like a break point for testing later on.

The above description assumes you are using the onboard LEDs and keyboard. If you use a serial terminal, the keystrokes are a bit different. On the KIM Uno, pressing [TAB] on the serial port makes you enter TTY mode. Read the KIM-1 user manual (link is provided on next page) for details of how to use it - and remember to have Caps Lock on at all times.

That's it. It used to be simple with computers, once upon a time! For further reference, read/use:

- KIM-1 User Manual: http://users.telenet.be/kim1-6502/6502/usrman.html#31
- First Book of KIM: users.telenet.be/kim1-6502/6502/fbok.html
- Andrew Jacobs' 6502 Reference web page www.obelisk.demon.co.uk/6502/reference.html holds all 6502 mnemonics, their functions and byte codes in the best possible way. Use your browsers Search function to get around the page.
- Mass:werk's online assembler www.masswerk.at/6502/assembler.html gets you from assembler to byte code very fast. In case you do not know 6502 byte codes by heart and do not care to change that.

2. Loading/saving to Eeprom - or use a 2nd K of RAM

The 1K EEPROM in the atMega is mapped as a second K of RAM. So any access to addresses 0x400-0x800 goes through to the EEPROM. You can either use it as extra RAM memory (it will be a bit slower, but not noticeably so for most things), or use it as a space to save your work. I.e, copy your program from RAM at 0200 to EEPROM starting at 0400. For writing to work, press the [RS] key for more than a second (or press '>' on the serial terminal) to toggle between R/O (default) and R/W Eeprom access. Write protection seemed useful because although there's at least a hundred thousand write cycles in the Eeprom, you can wear it out eventually. Theoretically.

3. Utility: Movit

This is the first of three crucial utility programs from the famous First Book of KIM by Jim Butterfield. Movit allows you to comfortably move sections of memory (for instance, 0200-0240 might contain your freshly developed program) to anywhere else (for instance, copy it to 0400-0440 where it will be saved into the eeprom. Do not forget to write-enable the eeprom by pressing ST for more than 1 second first).

At boot-up of the KIM Uno, movit is copied to the 64 bytes of RAM at **$1780**. You can overwrite it with your own code. It is also fully relocatable so you can move it anywhere. Movit is not a *relocator* - it does not adjust any adresses or jumps in the program you want to move. It just moves contents of a memory block to somewhere else.

> Enter original start adress of the program to move in $00D0 (LSB) and $00D1 (MSB),
> Enter original end adress of the program to move in $00D2 (LSB) and $00D3 (MSB),
> Enter new start adress of the program to move in $00D4 (LSB) and $00D5 (MSB),
> and press 1780 [GO].

4. Utility: Relocate

Relocate complements movit: it adjusts those bits in your code that contain calling addresses and jumps that would otherwise be wrong when you move a program in memory. Relocate is not a trivial program. KIM Uno stores it at **$0110**. That means it lives in the stack space of the 6502, which is fine because the monitor ROM only uses 8 bytes of the whole stack page. But if you write huge programs, you might overwrite Relocate. Which is fine, it is not needed for anything else.

The following is quoted from Jim Butterfield, the author, in the First Book of KIM:

Ever long for an assembler? Remember when you wrote that 300 byte program - and discovered that you'd forgotten one vital instruction in the middle? And to make room, you'd have to change all those branches, all those address... RELOCATE will fix up all those addresses and branches for you, whether you're opening out a program to fit in an extra instruction, closing up space you don't need, or just moving the whole thing someplace else.

RELOCATE doesn't move the data. It just fixes up the addresses before you make the move. It won't touch zero page addresses; you'll want them to stay the same. And be careful: it won't warn you if a branch instruction goes out of range.

You'll have to give RELOCATE a lot of information about your program:

1. Where your program starts. This is the first instruction in your whole program (including the part that doesn't move). RELOCATE has to look through your whole program, instruction by instruction, correcting addresses and branches where necessary. Be sure your program is a continuous series of instructions (don't mix data in; RELOCATE will take a data value of 10 as a BEL instruction and try to correct the branch address), and **place a dud instruction (FF) behind your last program instruction**. This tells RELOCATE where to stop. Place the **program start address in locations EA and EB**, low order first as usual. Don't forget the FF behind the last instruction; it doesn't matter if you temporarily wipe out a byte of data - you can always put it back later.
2. Where relocation starts, this is the first address in your program that you want to move. If you're moving the whole program, it will be the same as the program start address, above. This address is called the boundary. Place the **boundary address in locations EC and ED**, low order first.
3. How far you will want to relocate information above the boundary. This value is called the increment. For example, if you want to open up three more locations in your program, the increment will be 0003. If you want to close up four addresses, the increment will be FFFC (effectively, a negative number). Place the **increment value in locations E8 and E9**, low order first.
4. A page limit, above which relocation should be disabled. For example, if you're working on a program in the 0200 to 03FF range, your program might also address a timer or I/O registers, and might call subroutines in the monitor. You don't want these addresses relocated, even though they are above the boundary! So your page limit would be 17, since these addresses are all over 1700. On the other hand, if you have memory expansion and your program is at address 2000 and up, your page limit will need to be much higher. You'd normally set the page limit to FF, the highest page in memory. **Place the page limit in location E7**.

Now you're ready to go. Set RELOCATE's start address [0110], hit [GO] - and ZAP!-your addresses are fixed up. After the run, it's a good idea to check the address now in 00EA and 00EB - it should point at the FF at

the end of your program, confirming that the run went OK. Now you can move the program. If you have lots of memory to spare, you can write a general MOVE program and link it in to RELOCATE, so as to do the whole job in one shot. [this is why movit is included] Last note: the program terminates with a BRK instruction. Be sure your interrupt vector (at l7FE and 17FF) is set to KIM address 1C00 so that you get a valid "halt" [note: this is always done in the KIM Uno].

5. Utility: Branch

Branch is a small tool to calculate relative jumps. Start it through **01A5[GO]**. Then, key in the last two digits of a branch instruction address; then the last two digits of the address to which you are branching; and read off the relative branch address.

> Example:
> To calculate the branch value for a BEQ instruction located at $0226, needing to jump back to $0220: hit 26 (from 0226); 20 (from 0220) and the right offset, F8 shows up on the the display.

Keep in mind that the maximum "reach" of a branch instruction is 127 locations forward (7F) or 128 locations backward (80). If you want a forward branch, check that the calculated branch is In the range 0l to 7F. Similarly, be sure that a backward branch produces a value from 80 to FE. In either case, a value outside these limits means that your desired branch is out of reach.

The program must be stopped with the RS key.

6. Utility: Disassembler

Last but not least. Very much not least. Disassembler was written by Steve Wozniak and Allen Baum and published in the September 1976 edition of Dr. Dobbs. This KIM-1 port is from Bob Kurtz (1979) in *6502 User Notes #14*. In 505 bytes, Baum and Wozniak were able to write a full disassembler... seriously vintage bytes.

Store the address from which you want to start disassembling in **$00 (high byte, ie 02) and $01 (low byte**, ie 00). Obviously, given that this outputs text, you will need the serial terminal hooked up. Then enter **2000 [GO]** or, in serial TTY mode, type **2000<space>G**. The first 13 lines of disassembly roll by. Hit G or [GO] again to continue.

7. Utility: Floating Point Library

The fltpt65 floating point package resides at **$5000**. See the 'How to use 6502 Calculator' section if you just want to use it as a (more or less) normal calculator. But you can use this ROM in any KIM-1 program. To do so, load the operation (add, subtract, sin, cos, etc - see the Calculator section) in the A register and do a JSR $5000. Note that fltpt65 will use page 1 memory, erasing the relocate and branch programs. Should you want to use them afterwards, you need to press the reset button on the back of the Pro Mini.

8. Loading and saving programs to your PC

In serial mode (ie, Option 2, press [TAB] to go there), saving goes as follows: store the **end** address in 0x17F7 (upper byte, i.e. 02) and 0x17F8 (lower byte, i.e. FF). Now go to the address that is the **starting** address (0200 most likely). Press Q and a data file will be sent to the terminal. On the PC, you can then copy it, or capture it to a text file. For all the KIM knows, it has just created a paper tape... The command L allows you to upload a file into the KIM-1. Just press L, the KIM will wait for paper tape data so let your terminal program send the file. In Windows' HyperTerminal: Transfer->Send Text File. You may need to lower the speed at which the PC is sending bytes, HyperTerminal has a setting for that. A utility, kimpaper, exists to convert between binary files and the KIM's 'paper tape format'. See Appendix 1.

9. Keyboard templates

The KIM Uno special keys:

- Press **[ST]** for more than one second, and the write-protect on the eeprom is toggled on or off. On the serial terminal, this is the '>' key.
- Press **[SST]** for more than one second, and you toggle in/out of Calculator Mode. Which, as of September 1 2014, is a bad idea because you fall into test routines. But soon, you can then press C,D,E,F to view and edit the floating point registers of fltpt65. But that is another subject.

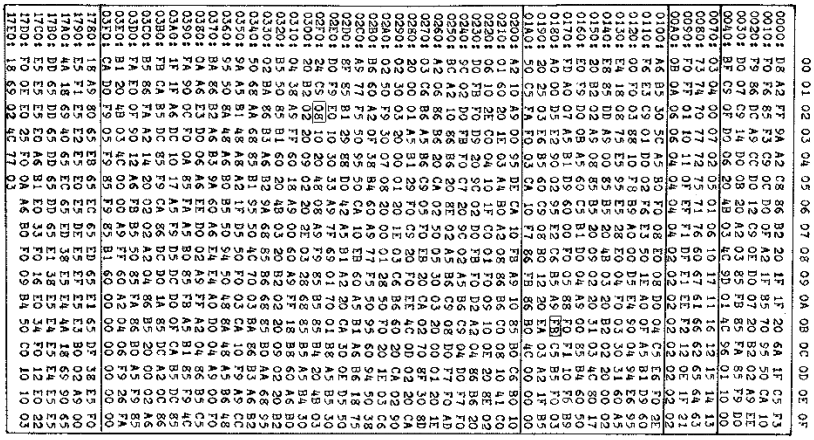
Other key mappings:

- Top line: Calculator Mode key functions
- Second line: Microchess key functions
- Bottom line: keystrokes used on the Serial port in Mode 1. (For Option 2, please refer to the original KIM-1 User Manual).

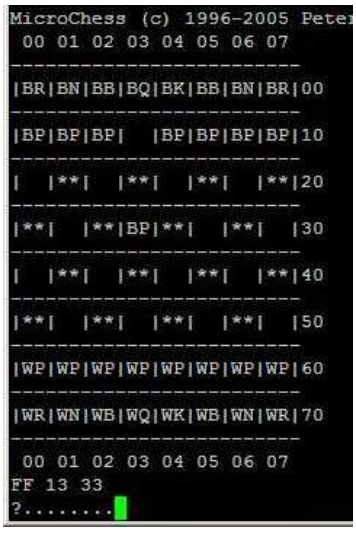
Correction to picture: in Calculator Mode, A is decimal point, B is 'E'.

Microchess

Peter Jennings published Microchess in December 1976. It became the Killer Application for the KIM-I... a full chess program in 924 bytes of 6502 code! Microchess was a defining moment in early microcomputer development - and also is a work of art, a coding masterpiece.



Jennings' own site (www.benlo.com/microchess) gives a fascinating account of how it came to be. Imagine having no assembler, no software tools and no information other than the manual and data books... entering hand-assembled hex codes into the KIM and creating a chess program in just 924 bytes. That begins to tell how awesome Microchess really is.



This version of Microchess is the one found on Peter Jennings' site. It adds a routine at its end that prints out its board on the serial port. Playing on the KIM Uno without a serial terminal will create a right proper pioneering KIM-I kind of feel... Meaning you get to see the moves on the LED display, but for an overview of the board you still need the serial port hooked up. Unless you have an actual chess board on which you keep track.

Command Summary:

Microchess was recompiled to run from a ROM at $C000. So enter C000 [GO] to start. Enter Q (serial port) or [RS] (KIM keypad) to go back to the KIM monitor.

KIM Key (Serial)	KIM Key (Serial)
C (C) Clear Board	1-7 (1-7) Keys to enter move
E (E) Exchange sides	F (Retrn) Register move
PC (P) Play	
*4 (L) Load Board	*5 (S) Save Board
+ (W) Toggle Blitz (fast)/Normal play (100sec/move)	

Gameplay is intuitive if you think of it this way:

- Start with C-E-PC (Clear-Exchange Sides-Play) to ask Microchess for a move.
- If you are lazy, you keep doing E-PC-E-PC-E-PC to let Microchess play itself.
- Or, you enter your own moves by entering (example - see below) 1333 F, followed by PC to let Microchess make its next move in response.

Some extra functions have been added, based on suggestions in the Microchess manual. These are implemented as interventions on top of the 6502 emulator. Rest assured, no vintage bytes from within Microchess have been hurt in the process. You're running 100% original 6502 Microchess :)

Blitz mode: [+], or W, toggles between normal speed (approx. 100 sec per move) and Blitz mode (approx. 10 sec. per move, the default setting). Note that on the serial port, a dot is printed for every step taken during Microchess' next move calculation. On the KIM display, you see nothing until Microchess is done with its calculations - which is how it was originally...

Save Board: You can save your game using the EEPROM in the Arduino. This simply saves the current chess board so you can continue playing at a later time. Press S (serial) or ?? and then enter the slot number you want to save to - a number from 0 to 9. Confirm by hitting return (serial) or ?? and the board is saved. **Load board**: done by L (serial) or ??. This retrieves a saved game and allows you to continue to play it.

Quick Guide: How to Play Microchess

Enter C000 [GO] to run Microchess. If you want Microchess to play its best, leave Blitz mode by pressing [+], or W (that's shift-W) on the serial terminal. But the 100 second wait for moves to be calculated is probably annoying. The board print-out confirms by a line showing >N<.

```
1. Initialise
=============
Enter C for Clear. Actually, at any time, hit C to reset the game.
--> The board is reprinted & bottom shows CC CC CC to confirm.

2. Let the computer play against itself
=======================================
Enter E to make the computer take the opposite side of the board.
--> The bottom line shows EE EE EE to confirm you swapped chairs.

Enter PC (P on serial port) to make the computer Play its move.
--> Deep thought happens. Every few seconds, a dot appears on the serial
    terminal to indicate one more possible move has been thought through.
    But the LEDs go dark for this period - that's how it is supposed to be.
--> Typically, you'll see 30 or so dots before the computer is done.
    In the default Blitz mode, it takes about 15 seconds on a real KIM-I,
    more or less the same on the Arduino. In Normal mode, about 100 seconds.

When the best move has been decided, MicroChess prints out the board again.
--> The bottom line shows its move in 3 hex numbers, also shown on the KIM Uno LEDs.
        Hex number 1, left digit: 0 if a Black piece was moved, 1 for White
        Hex number 1, right digit: Tells you what piece this was:
             ------------------------------------------------------------
            | 0 - KING      | 4 - King Bishop  | B - K R Pawn | C - K B Pawn |
            | 1 - Queen     | 5 - Queen Bishop | 9 - Q R Pawn | D - Q B Pawn |
            | 2 - King Rook | 6 - King Knight  | A - K N Pawn | E - Q Pawn   |
            | 3 - Queen Rook| 7 - Queen Knight | B - Q N Pawn | F - K Pawn   |
             ------------------------------------------------------------
        Hex number 2 is the FROM square (row, column).
        Hex number 3 is the TO square (row, column).

So you can let the computer play both sides by hitting E, then PC, E again, PC...
```

```
3. Entering your own move
=========================
Instead of hitting PC (or P on serial port), you can enter the 2-digit FROM square and
then the 2-digit TO square. After every digit, you'll see the board reprinted. But focus
on the bottom line: the digit you press "rolls in" to the bottom line's Numbers 2 and 3
from right to left. This is a remnant of how the KIM-I uses its onboard LEDs.

After four digits, the move is defined. MicroChess shows the piece involved in Hex Number
1: first digit is 0 for white, 1 for black. Second digit is as per the table above.

Once your move is complete, hit F (or Return on serial port) to register your move.
--> FF appears in Hex Number 1: you have now moved the piece,
    so FF is there to show that the From square is now empty.

After you've registered the move you can still undo it. Just correct the wrong move by
entering four more digits in a correcting move, so that the board looks like you intended
it to.
Indeed - this is the fundamental point to grasp: MicroChess does not check what you're
doing when you move pieces around. You can move its pieces as well as your own. All you
are doing here is rearranging the board for the next round of Microchess Deep Thought.
You can make a normal move, or shift stuff around on the board as you wish to create a
new situation that MicroChess should play against next. It's a feature, not a limitation!
--> Hit PC to make MicroChess Play its next move.

4. Special moves:
=================
Castling: just move the two pieces and hit Return after each one to register them both.
En passant: break the move up in two moves. Mid-point being on the piece you strike out.
Queening pawns: yes. Well. Remember which of your pawns has been Queened and give it the
according moves afterwards. MicroChess will not Queen on its side. (It can be done by
leaving the program and manipulating its memory through the KIM Monitor).
```

Further reading on MicroChess

The original manual is here:
http://users.telenet.be/kim1-6502/microchess/microchess.html.

Original source code of Microchess for serial terminal:
http://benlo.com/files/Microchess6502.txt.

To translate between Kim-1 keys and the keys on the serial terminal:

Kim-1 key	Arduino Serial port
C	C
E	E
PC	P
F	Return

Just to make sure proper credit is given: A thank-you to the genius of Peter Jennings. It is used here with his permission, but please note that Microchess is not open source code.

6502 Programmable Calculator

This page describes the calculator functions of the KIM Uno. Software archaeology is nice; 6502 programming is fun and interesting. But you have got to wonder *what* you are actually going to program, regularly, with something as obsolete as a KIM-1 in the 21st century...

The one useful role such technology still can perform well is to be a calculator. So the idea is to provide both a simple-to-use normal calculator (with a geeky bent) and a 6502 programmable calculator, using the KIM-1 environment with minimal extensions.

Quick Start for Simple Calculator Mode: let's add 123 + 456.

```
Press SST for 1 second to enter Calculator Mode
Press C, 123, D, 456, F, 00, GO, PC --> Your answer is 579
```

More precisely, to use the KIM Uno as just a simple calculator:

- Press **SST** for a second. The display flashes & keyboard changes to Calculator Mode.
- Press **C** to enter your first number. The following keys are redefined:
    - A--> decimal point
    - B --> use scientific notation: adds 'E' as in Exponent. So 1A25E3 is 1.25E3 is 125.
    - + --> toggle to minus. Yes, the plus key means minus. I found that funny but intuitive.
    - GO --> Enter the number; or ST --> Cancel. Both return you to the KIM display.
      Or, instead of pressing GO, alternatively:...
- Press **D** to enter your second number.
- Then press **F** to enter the operation number (calculation) you want to do.
    - 00 is add, 01 subtract, 02 multiply. See the table below for all 38 operations.
- Press **GO** straight after and the fltpt65 library will be run.
- Press **PC** to view the result.

Keystroke Summary for Simple Calculator Mode

```
Entering & Viewing numbers:

 C - Enter W1 value  [AD] - View W1
 D - Enter W2 value  [DA] - View W2
                     [PC] - View W3
result
 F - Enter desired operation
```

```
Entering numbers after pressing C or D:

 A - Decimal point
 B - Add exponent for scientific
notation
 + - Toggle sign

 GO- Store number
 ST- Cancel
 D - Store&continue entering second nr
 F - Store&continue entering operation
```

```
Operations after pressing F:

add: 00   sin: 10   asec: 20   asinh: 30
sub: 01   cos: 11   acot: 21   acosh: 31
mul: 02   tan: 12   loge: 22   atanh: 32
div: 03   csc: 13   exp:  23   acsch: 33
sqrt:04   sec: 14   sinh: 24   asech: 34
inv: 05   cot: 15   cosh: 25   acoth: 35
int: 06   asin:16   tanh: 26   log2:  36
frac:07   acos:17   csch: 27   log10: 37
abs: 08   atan:18   sech: 28   pow:   38
chs: 09   acsc:19   coth: 29


After entering the two-digit number,
the KIM's PC is set for you, so
➔  press [GO] to run the calculation,
➔  then press [PC] to view result in W3
```

Technical interlude: fltpt65

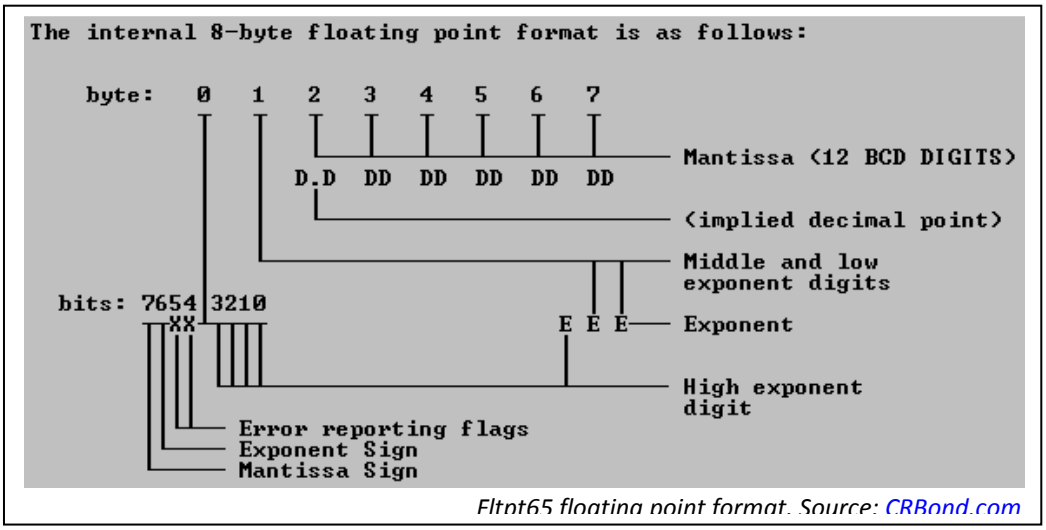
If you think of early microcomputers as glorified calculators, it's surprising to see how hard it is to do floating point math on a CPU. Some of the very earliest programming on microprocessors – even before operating systems arrived - was thus to produce good math routines. In 1976, MOS Technology published KIMATH, and Roy Rankin and Steve Wozniak published a competing math library in Dr. Dobb's Journal. In recent years, Charles R Bond has created fltpt65, which is probably the last word in 6502 math programming. This is the code used in the KIM Uno. It's placed in a ROM starting at $5000.

Fltpt65 requires you to:

- enter the two **numbers** you want to work with in its **W1 and W2 registers**,
- enter the **operation** you want it to perform **in the 6502's A register**,
- Then, you call the program at $5000, and fltpt65 will provide the **result** in its **W3 register**.
  There also is a W4, which you can use as an extra calculator memory. The four registers are actually 8-byte locations in memory, starting at $0360.

The above is simple enough. But W1 and W2 are encoded floating point numbers, not just single-byte integers, so they are hard to handle. Each W register is an array of 8 bytes that (ignore the following unless you care) represents a floating point number, BCD encoded, with 12 mantissa digits and 3 exponent digits supporting a maximum value of +9.999999999 E+999.

The picture shows the format. But the point is, it's hard to use, so the Uno has to provide a user interface.



Fltpt65 floating point format. Source: CRBond.com

Simple access to the W registers

To make life easy, the KIM Uno provides keystrokes that allow you to simply enter and read numbers in the W registers, forgetting about their underlying complexity. That is - basically - what is called the Calculator mode: you briefly step out of the emulated KIM-1 to enter or display, one of the W registers. Once you've entered the number, or you've seen it, you're dropped back into the KIM-1 so fltpt65 can do its calculations.

Whilst you're looking at W registers, you'll notice that you have 7 LED digits and, of course, decimal points between them. The moment you drop back into the KIM, you revert to the original 6-digit KIM-1 display.

Quick Start: The KIM as a 6502 Programmable Calculator

Leave simple Calculator Mode if you are in it (press SST for >1 sec again). Try the built-in demo program at $0210. It asks you to enter two numbers and shows you the product of them. Then, it asks you for an integer number, used as a yes/no question. If you enter 00, the program ends by showing the last result again. If you enter 01, it shows the square root of the previous result.

```
Enter 0210 [GO]                (to start demo program)
Enter 4 [GO], then 5 [GO]      (input numbers into W1 & 2)
View number, then press [GO]   (4*5=20.000)
Enter 01 [GO]                  (01 = yes, take square root)
View number, then press [GO]   (sqrt(20) = 4.472135)
```

6502 Programmable Calculator: using fltpt65 in small KIM-1 programs

So you do not need to enter Calculator Mode to run the fltpt65 subroutines. You can just call them in any program you like. Just make sure that W1 and W2 contain the floating point numbers you want to work with. And there, the function calls demonstrated just now will come in very handy.

This makes the KIM Uno a pretty decent programmable calculator, which you code in 6502 instructions. The code can be **very** simple; see the example program stored at $0210:

```
      JSR $70A1    ; ask user for W1 input
      JSR $70A2    ; ask user for W2 input
      LDA #$02     ; want to multiply W1 * W2
      JSR $5000    ; execute calculation
      JSR $70D3    ; display the W3 result register
      JSR $70AA    ; ask user for integer value in A register
      BEQ SKIP     ; skip sqrt calculation if user entered 0
      JSR $7031    ; copy result of multiplication in W3 to W1
      LDA #$04     ; take square root of W1
      JSR $5000    ; execute calculation
SKIP: JSR $70D3    ; display W3
      BRK          ; end
```

The function call addresses are kind of logical if you think of them this way:

- Collectively called the 7000 calls;
- The **(A)sk a number** calls are 7-0-**A**-*Register number*
- The **(D)isplay a number** calls are 7-0-**D**-*Register number*
- The **copy** calls are the 7-**0** series (7-0-*X*-to-*Y*)
- The **swap** calls are the 7-**1** series (7-1-*X*-with-*Y*)

```
$7XXX Function calls for 6502 Programs:

-$70Ax: Ask user for input-------------------------
 $70A1/2/3/4: Ask user to enter number in W1/2/3/4
 $70AA: Ask user to enter integer into A register

-$70Dx: Display number----------------------------
 $70D1/2/3/4: Display number in W1/2/3/4

-$70xy: Copy number between registers-------------
 $70xy: Copy Wx to Wy

-$71xy: Swap numbers in registers-----------------
```

A few closing notes:

- Remember you can store code in Eeprom (1K starting from $0400) so nothing gets lost if you power off.

- When programming, don't forget that you can use **disasm** to view assembly code on the serial port. Put the address of your program in 0000 and 0001, and do 2000 [GO].
  At least until you know the opcodes for LDA, JSR and BEQ and maybe a few more by heart, disassembled 6502 is a lot easier on the eye.

- The Mass:werk online assembler is a comfortable assembler if you need one. Lastly, ou may want to try the **branch** calculator at $01A5 if you use the BEQ opcode and want to calculate relative offsets.

- Fltpt65 uses page 3 ($300-3AF) and Page 1 ($100-1A0) of the KIM-1 RAM as its work registers. Using fltpt65 means any data sitting there will be destroyed. You still have the 256 bytes of page 2 ($200-2FF) available to you, plus the 2 * 64 bytes stashed in the RIOTs. That amount of memory was enough to put men on the moon... You also have the 1K eeprom from $400-$800. That'll bring you to Mars. Easily.

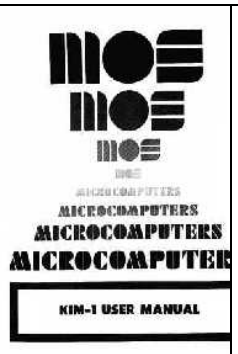
Appendix 1: Useful links

KIM Uno materials

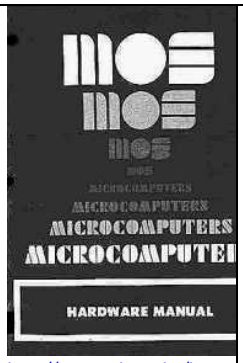
- Project home page with all hard/software downloads: http://obsolescence.wix.com/obsolescence#!kim-uno-details/c1alo
- Seeedstudio: recommended PCB manufacturer. The ZIP archive with the Gerber PCB design files can be submitted to them, for about $30 they will send 5 freshly minted PCBs to you. http://www.seeedstudio.com/service/index.php?r=pcb

KIM-1 documentation & books

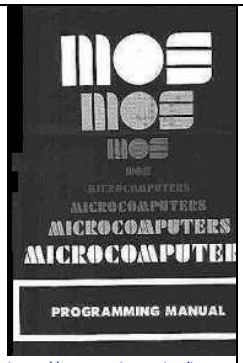
- Manuals



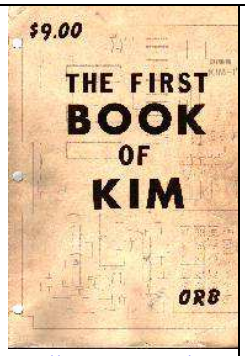
http://users.telenet.be/kim1-6502/6502/usrman.html



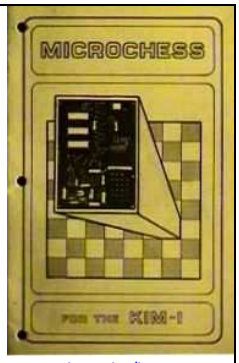
http://users.telenet.be/kim1-6502/6502/hwman.html



http://users.telenet.be/kim1-6502/6502/proman.html



http://users.telenet.be/kim1-6502/6502/fbok.html



users.telenet.be/kim1-6502/microchess/microchess.html

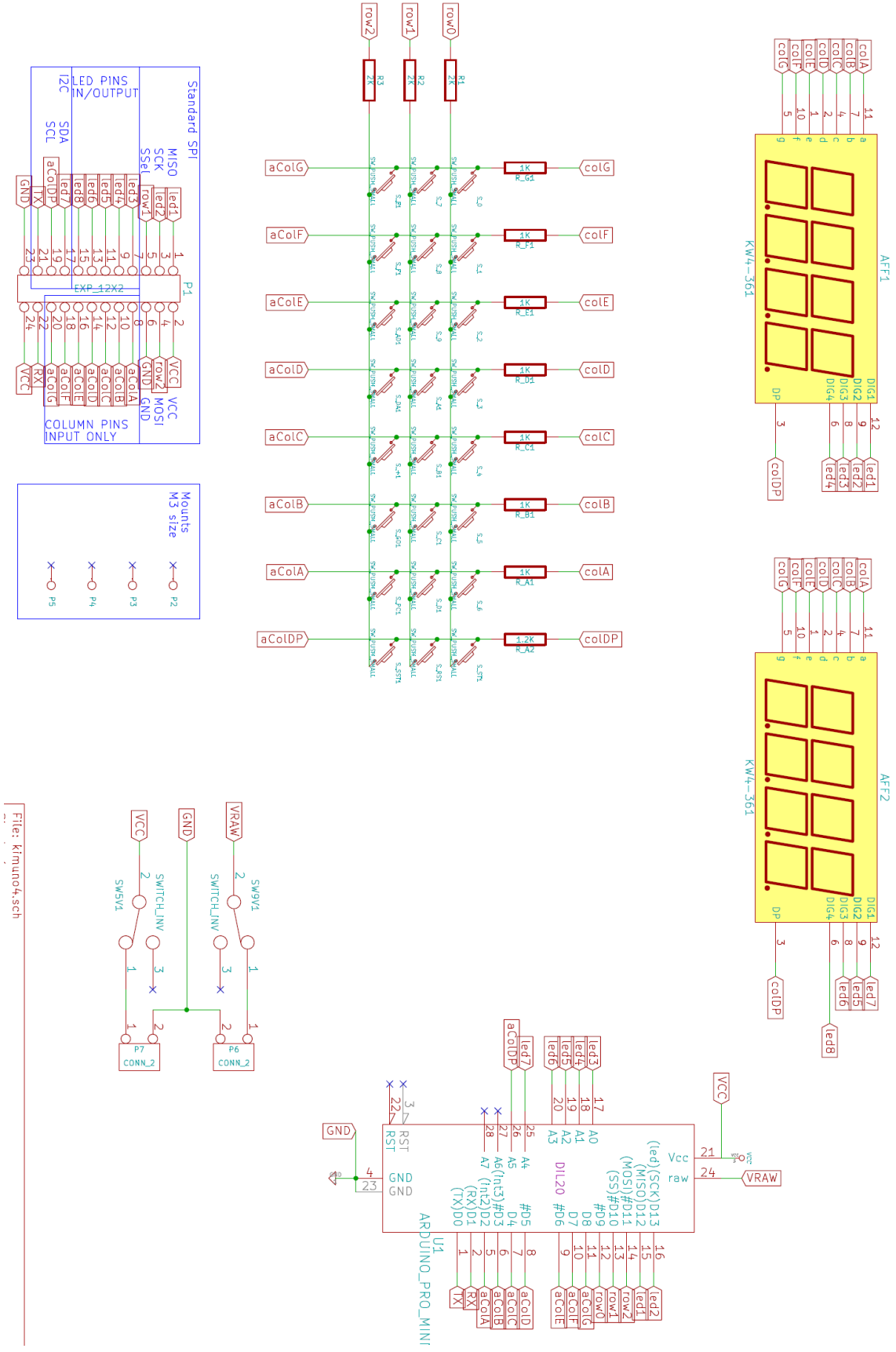
- KIM Rom disassembled sources with comments
  - http://users.telenet.be/kim1-6502/6502/kim.txt
  - http://www.baltissen.org/htm/src_kim.htm

- KIM-1 User Notes: http://6502.org/documents/publications/6502notes/

Utilities

- KIMPAPER – converts files between binary & serial port 'paper tape format': http://retro.hansotten.nl/index.php?page=micro-kim (look for KIMPAPER)
- puTTY – a popular terminal program. Set to 9600bps for the Uno. http://www.putty.org/
- HyperTerm – a terminal program that comes with older versions of Windows.

Appendix 2: Schematics



Appendix 3: 6502 opcodes

More in-depth reference cards are on the following pages. But the picture below should print out at 6x9 cm, fitting nicely inside a cover for the KIM Uno:

ADC	Imm	69	CMP	ZP,X	D5	LDA	Abs	AD	ROR	Acc	6A
ADC	ZP	65	CMP	Abs	CD	LDA	Abs,X	BD	ROR	ZP	66
ADC	ZP,X	75	CMP	Abs,X	DD	LDA	Abs,Y	B9	ROR	ZP,X	76
ADC	Abs	6D	CMP	Abs,Y	D9	LDA	(Indir,X)	A1	ROR	Abs	6E
ADC	Abs,X	7D	CMP	(Indir,X)	C1	LDA	(Indir),Y	B1	ROR	Abs,X	7E
ADC	Abs,Y	79	CMP	(Indir),Y	D1	LDX	ZP	A6	RTI		40
ADC	(Indir,X)	61	CPX	Imm	E0	LDX	ZP,Y	B6	RTS		60
ADC	(Indir),Y	71	CPX	ZP	E4	LDX	Abs	AE	SBC	Imm	E9
AND	Imm	29	CPX	Abs	EC	LDX	Abs,Y	BE	SBC	ZP	E5
AND	ZP	25	CPY	Imm	C0	LDX	Imm	A2	SBC	ZP,X	F5
AND	ZP,X	35	CPY	ZP	C4	LDY	Imm	A0	SBC	Abs	ED
AND	Abs	2D	CPY	Abs	CC	LDY	ZP	A4	SBC	Abs,X	FD
AND	Abs,X	3D	DEC	ZP	C6	LDY	ZP,X	B4	SBC	Abs,Y	F9
AND	Abs,Y	39	DEC	ZP,X	D6	LDY	Abs	AC	SBC	(Indir,X)	E1
AND	(Indir,X)	21	DEC	Abs	CE	LDY	Abs,X	BC	SBC	(Indir),Y	F1
AND	(Indir),Y	31	DEC	Abs,X	DE	LSR	Acc	4A	SEC		38
ASL	Acc	0A	DEX		CA	LSR	ZP	46	SED		F8
ASL	ZP	6	DEY		88	LSR	ZP,X	56	SEI		78
ASL	ZP,X	16	EOR	Imm	49	LSR	Abs	4E	STA	ZP	85
ASL	Abs	0E	EOR	ZP	45	LSR	Abs,X	5E	STA	ZP,X	95
ASL	Abs,X	1E	EOR	ZP,X	55	NOP		EA	STA	Abs	8D
BCC		90	EOR	Abs	4D	ORA	Imm	9	STA	Abs,X	9D
BCS		B0	EOR	Abs,X	5D	ORA	ZP	5	STA	Abs,Y	99
BEQ		F0	EOR	Abs,Y	59	ORA	ZP,X	15	STA	(Indir,X)	81
BIT	ZP	24	EOR	(Indir,X)	41	ORA	Abs	0D	STA	(Indir),Y	91
BIT	Abs	2C	EOR	(Indir),Y	51	ORA	Abs,X	1D	STX	ZP	86
BMI		30	INC	ZP	E6	ORA	Abs,Y	19	STX	ZP,Y	96
BNE		D0	INC	ZP,X	F6	ORA	(Indir,X)	1	STX	Abs	8E
BPL		10	INC	Abs	EE	ORA	(Indir),Y	11	STY	ZP	84
BRK		0	INC	Abs,X	FE	PHA		48	STY	ZP,X	94
BVC		50	INX		E8	PHP		8	STY	Abs	8C
BVS		70	INY		C8	PLA		68	TAX		AA
CLC		18	JMP	Indir	6C	PLP		28	TAY		A8
CLD		D8	JMP	Abs	4C	ROL	Acc	2A	TSX		BA
CLI		58	JSR		20	ROL	ZP	26	TXA		8A
CLV		B8	LDA	Imm	A9	ROL	ZP,X	36	TXS		9A
CMP	Imm	C9	LDA	ZP	A5	ROL	Abs	2E	TYA		98
CMP	ZP	C5	LDA	ZP,X	B5	ROL	Abs,X	3E			

MICRO LOGIC CORP.
HACKENSACK, NJ

6502 (65XX)

MICROPROCESSOR INSTANT REFERENCE CARD

MICRO® CHART

Hex to Instruction Conversion

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F	
0-	BRK	ORA (n,X)				ORA n	ASL n		PHP	ORA #n	ASL A			ORA nn	ASL nn		0-
1-	BPL n	ORA (n),Y				ORA n,X	ASL n,X		CLC	ORA nn,Y				ORA nn,X	ASL nn,X		1-
2-	JSR nn	AND (n,X)			BIT n	AND n	ROL n		PLP	AND #n	ROL A		BIT nn	AND nn	ROL nn		2-
3-	BMI n	AND (n),Y				AND n,X	ROL n,X		SEC	AND nn,Y				AND nn,X	ROL nn,X		3-
4-	RTI	EOR (n,X)				EOR n	LSR n		PHA	EOR #n	LSR A		JMP nn	EOR nn	LSR nn		4-
5-	BVC n	EOR (n),Y				EOR n,X	LSR n,X		CLI	EOR nn,Y				EOR nn,X	LSR nn,X		5-
6-	RTS	ADC (n,X)				ADC n	ROR n		PLA	ADC #n	ROR A		JMP (nn)	ADC nn	ROR nn		6-
7-	BVS n	ADC (n),Y				ADC n,X	ROR n,X		SEI	ADC nn,Y				ADC nn,X	ROR nn,X		7-
8-		STA (n,X)			STY n	STA n	STX n		DEY		TXA		STY nn	STA nn	STX nn		8-
9-	BCC n	STA (n),Y			STY n,X	STA n,X	STX n,Y		TYA	STA nn,Y	TXS			STA nn,X			9-
A-	LDY #n	LDA (n,X)	LDX #n		LDY n	LDA n	LDX n		TAY	LDA #n	TAX		LDY nn	LDA nn	LDX nn		A-
B-	BCS n	LDA (n),Y			LDY n,X	LDA n,X	LDX n,Y		CLV	LDA nn,Y	TSX		LDY nn,X	LDA nn,X	LDX nn,Y		B-
C-	CPY #n	CMP (n,X)			CPY n	CMP n	DEC n		INY	CMP #n	DEX		CPY nn	CMP nn	DEC nn		C-
D-	BNE n	CMP (n),Y				CMP n,X	DEC n,X		CLD	CMP nn,Y				CMP nn,X	DEC nn,X		D-
E-	CPX #n	SBC (n,X)			CPX n	SBC n	INC n		INX	SBC #n	NOP		CPX nn	SBC nn	INC nn		E-
F-	BEQ n	SBC (n),Y				SBC n,X	INC n,X		SED	SBC nn,Y				SBC nn,X	INC nn,X		F-
	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F	

Memory Map

ZERO PAGE	0000
	00FF
DATA & STACK*	0100
	01FF
RAM I/O ROM	0200
	FFF9
NMI VECTOR	FFFA&B
RES VECTOR	FFFC&D
IRQ VECTOR	FFFE&F

*In systems with < 512 bytes of RAM the hardware can ignore signal AB8, moving stack into page zero.

Status Flags

MSB LSB

N	V	-	B	D	I	Z	C

N=negative result
V=overflow
B=BRK instruction
D=decimal mode
I=IRQ disable
Z=zero result
C=carry=borrow

Note: above is true when flag = 1.

Overflow normally signifies signed arithmetic result is out of range.

When D=1, only ADC and SBC use decimal (BCD) arithmetic.

Effect on Flags

	N V - B D I Z C
ADC	N V - - - - Z C (1)
AND	N - - - - - Z -
ASL	N - - - - - Z C
BIT	N V - - - - Z - (2)
BRK	- - - 1 - 1 - -
CLC	- - - - - - - 0
CLD	- - - - 0 - - -
CLI	- - - - - 0 - -
CLV	- 0 - - - - - -
CMP	N - - - - - Z C
CPX	N - - - - - Z C
CPY	N - - - - - Z C
DEC	N - - - - - Z -
DEX	N - - - - - Z -
DEY	N - - - - - Z -
EOR	N - - - - - Z -
INC	N - - - - - Z -
INX	N - - - - - Z -
INY	N - - - - - Z -
LDA	N - - - - - Z -
LDX	N - - - - - Z -
LDY	N - - - - - Z -
LSR	0 - - - - - Z C
ORA	N - - - - - Z -
PLA	N - - - - - Z -
PLP	N V - B D I Z C
ROL	N - - - - - Z C
ROR	N - - - - - Z C
RTI	N V - B D I Z C
SBC	N V - - - - Z C (3)
SEC	- - - - - - - 1
SED	- - - - 1 - - -
SEI	- - - - - 1 - -
TAX	N - - - - - Z -
TAY	N - - - - - Z -
TSX	N - - - - - Z -
TXA	N - - - - - Z -
TYA	N - - - - - Z -

(1) If in decimal mode Z flag is invalid.

(2) N = data bit 7
V = data bit 6
Z = AND result

(3) C̄ = borrow

Note: unlisted instructions have no effect on flags.

Interrupts

IRQ is low level sensitive.
NMI is falling edge sensitive.
Reset sets I=1.
Interrupts are processed by:

1. Push PC of unexecuted instruction.
2. Push P.
3. I=1.
4. Jump via appropriate vector.

Addressing Modes

Note: Full 2 byte addresses in code, stack, and data areas are stored low byte followed by high byte. Thus, in hex, JMP $1234 is: 4C 34 12.

FORM	ADDRESSING	DESCRIPTION
nn	Absolute	Location nn holds data.
nn,X	Absolute X	Location nn+X holds data.
nn,Y	Absolute Y	Location nn+Y holds data.
A	Accumulator	Accumulator holds data.
#n	Immediate	n is data.
(n,X)	Ind X	Location n+X and next of page 0 hold address of data.*,**
(n), Y	Ind Y	Address of data is Y + address held by location n and next of page 0.**
(nn)	Indirect	Location nn and next hold adddress to jump to.**
n	Relative	Address to jump to is n + address of next instruction, with n treated as a signed number.
n	Zero Page	Location n of page 0 holds data.
n,X	Zero Page X	Location n+X of page 0 holds data.
n,Y	Zero Page Y	Location n+Y of page 0 holds data.

*n+X is computed discarding any carry.
**2 bytes must not cross page boundary.

ASCII Character Set

LSD \ MSD	0 000	1 001	2 010	3 011	4 100	5 101	6 110	7 111
0	0000	NUL	DLE	SP	0	@	P	'	p
1	0001	SOH	DC1	!	1	A	Q	a	q
2	0010	STX	DC2	"	2	B	R	b	r
3	0011	ETX	DC3	#	3	C	S	c	s
4	0100	EOT	DC4	$	4	D	T	d	t
5	0101	ENQ	NAK	%	5	E	U	e	u
6	0110	ACK	SYN	&	6	F	V	f	v
7	0111	BEL	ETB	'	7	G	W	g	w
8	1000	BS	CAN	(	8	H	X	h	x
9	1001	HT	EM	)	9	I	Y	i	y
A	1010	LF	SUB	*	:	J	Z	j	z
B	1011	VT	ESC	+	;	K	[	k	{
C	1100	FF	FS	,	<	L	\	l	|
D	1101	CR	GS	-	=	M	]	m	}
E	1110	SO	RS	.	>	N	!	n	~
F	1111	SI	US	/	?	O	←	o	DEL

Miscellaneous

S points to next free byte of stack.

Stack push decrements S.

In pushing PC, high byte is pushed first.

Pre 6/76 chips have no ROR instruction.

65XX is a totally software compatible family.

This card is based on specifications from MOS Technology, Inc.

Registers

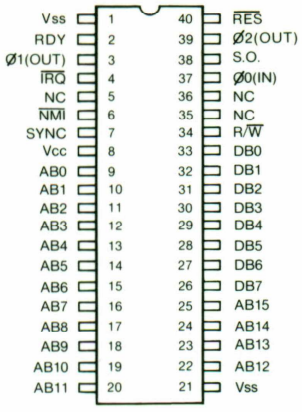
A	ACCUMULATOR
Y	Y INDEX REG
X	X INDEX REG
PC	PROGRAM COUNTER
S	STACK PNTR
P	FLAGS

A, Y, X, S, P - 1 byte.
Only PC is 2 bytes.

Hex and Decimal Conversion

LSD →	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F	
0	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	0
1	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	1
2	32	33	34	35	36	37	38	39	40	41	42	43	44	45	46	47	2
3	48	49	50	51	52	53	54	55	56	57	58	59	60	61	62	63	3
4	64	65	66	67	68	69	70	71	72	73	74	75	76	77	78	79	4
5	80	81	82	83	84	85	86	87	88	89	90	91	92	93	94	95	5
6	96	97	98	99	100	101	102	103	104	105	106	107	108	109	110	111	6
7	112	113	114	115	116	117	118	119	120	121	122	123	124	125	126	127	7
8	128	129	130	131	132	133	134	135	136	137	138	139	140	141	142	143	8
9	144	145	146	147	148	149	150	151	152	153	154	155	156	157	158	159	9
A	160	161	162	163	164	165	166	167	168	169	170	171	172	173	174	175	A
B	176	177	178	179	180	181	182	183	184	185	186	187	188	189	190	191	B
C	192	193	194	195	196	197	198	199	200	201	202	203	204	205	206	207	C
D	208	209	210	211	212	213	214	215	216	217	218	219	220	221	222	223	D
E	224	225	226	227	228	229	230	231	232	233	234	235	236	237	238	239	E
F	240	241	242	243	244	245	246	247	248	249	250	251	252	253	254	255	F
	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F	

6502 Pins



Abbreviations

B = number of Bytes
C = number of Cycles. also Carry.

n = 1 byte quantity
nn = 2 byte quantity

IRQ = Interrupt ReQuest
NMI = Non Maskable Interrupt
RES = RESet
XOR = eXclusive OR
(00►0 01►1 10►1 11►0)

A,P,S,X,Y,PC=see "Registers"
N,V,B,D,I,Z,C = see "Status Flags"
.#$@%'(); = see "Assembler Symbols"

Unsigned Comparisons

example: CMP # n

A < n	BCC YES
A = n	BEQ YES
A > n	BCC NO BNE YES
A ≥ n	BCS YES
A ≠ n	BNE YES
A ≤ n	BCC YES BEQ YES

YES represents label for code to be executed if condition is true. For > & ≤, test requires both instructions.

Internally, A-n is computed to determine N,Z,C flags.

100% PLASTIC

MICRO CHARTS: Z80, 6502-65XX, 8080-8085, 8086-8088, 8048 Family, 54/7400 TTL pinouts, BASIC Algorithms, Wordstar, Electronic Components, Sampling Statistics, C Language.

INSTANT ACCESS

Inexpensive plastic MICRO CHART references are easily purchased from leading dealers. You can also send a check, bearing your address on front and title(s) you want on back, to Micro Logic, POB 174, Dept C, Hackensack, NJ 07602

MICRO LOGIC CORP.
HACKENSACK, NJ

6502 (65XX)

MICROPROCESSOR INSTANT REFERENCE CARD

MICRO® CHART

INSTRUCTION SET

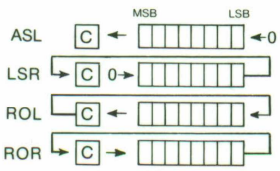
	INSTRUCTION	OP	C	B	DESCRIPTION	ADDRESSING
A	ADC #n	69	2	2	Add with carry to A	Immediate
	ADC nn	6D	4	3	Add with carry to A	Absolute
	ADC n	65	3	2	Add with carry to A	Zero Page
	ADC (n,X)	61	6	2	Add with carry to A	Ind X
	ADC (n),Y	71	5+	2	Add with carry to A	Ind Y
	ADC n,X	75	4	2	Add with carry to A	Zero Page X
	ADC nn,X	7D	4+	3	Add with carry to A	Absolute X
	ADC nn,Y	79	4+	3	Add with carry to A	Absolute Y
	AND #n	29	2	2	AND to A	Immediate
	AND nn	2D	4	3	AND to A	Absolute
	AND n	25	3	2	AND to A	Zero Page
	AND (n,X)	21	6	2	AND to A	Ind X
	AND (n),Y	31	5+	2	AND to A	Ind Y
	AND n,X	35	4	2	AND to A	Zero Page X
	AND nn,X	3D	4+	3	AND to A	Absolute X
	AND nn,Y	39	4+	3	AND to A	Absolute Y
	ASL nn	0E	6	3	Arithmetic shift left	Absolute
	ASL n	06	5	2	Arithmetic shift left	Zero Page
	ASL A	0A	2	1	Arithmetic shift left	Accumulator
	ASL n,X	16	6	2	Arithmetic shift left	Zero Page X
	ASL nn,X	1E	7	3	Arithmetic shift left	Absolute X
B	BCC n	90	2+	2	Branch if carry clear (C=0)	Relative
	BCS n	B0	2+	2	Branch if carry set (C=1)	Relative
	BEQ n	F0	2+	2	Branch if equal (Z=1)	Relative
	BNE n	D0	2+	2	Branch if not equal (Z=0)	Relative
	BMI n	30	2+	2	Branch if minus (N=1)	Relative
	BPL n	10	2+	2	Branch if plus (N=0)	Relative
	BVC n	50	2+	2	Branch if ovfl clear (V=0)	Relative
	BVS n	70	2+	2	Branch if ovfl set (V=1)	Relative
	BIT nn	2C	4	3	AND with A (A unchanged)	Absolute
	BIT n	24	3	2	AND with A (A unchanged)	Zero Page
	BRK	00	7	1	Break (force interrupt)	None
C	CLC	18	2	1	Clear carry	None
	CLD	D8	2	1	Clear decimal mode	None
	CLI	58	2	1	Clear IRQ disable	None
	CLV	B8	2	1	Clear overflow	None
	CMP #n	C9	2	2	Compare with A	Immediate
	CMP nn	CD	4	3	Compare with A	Absolute
	CMP n	C5	3	2	Compare with A	Zero Page
	CMP (n,X)	C1	6	2	Compare with A	Ind X
	CMP (n),Y	D1	5+	2	Compare with A	Ind Y
	CMP n,X	D5	4	2	Compare with A	Zero Page X
	CMP nn,X	DD	4+	3	Compare with A	Absolute X
	CMP nn,Y	D9	4+	3	Compare with A	Absolute Y
	CPX #n	E0	2	2	Compare with X	Immediate
	CPX nn	EC	4	3	Compare with X	Absolute
	CPX n	E4	3	2	Compare with X	Zero Page
	CPY #n	C0	2	2	Compare with Y	Immediate
	CPY nn	CC	4	3	Compare with Y	Absolute
	CPY n	C4	3	2	Compare with Y	Zero Page
D	DEC nn	CE	6	3	Decrement by one	Absolute
	DEC n	C6	5	2	Decrement by one	Zero Page
	DEC n,X	D6	6	2	Decrement by one	Zero Page X
	DEC nn,X	DE	7	3	Decrement by one	Absolute X
	DEX	CA	2	1	Decrement X by one	None
	DEY	88	2	1	Decrement Y by one	None
E	EOR #n	49	2	2	XOR to A	Immediate
	EOR nn	4D	4	3	XOR to A	Absolute
	EOR n	45	3	2	XOR to A	Zero Page
	EOR (n,X)	41	6	2	XOR to A	Ind X
	EOR (n),Y	51	5+	2	XOR to A	Ind Y
	EOR n,X	55	4	2	XOR to A	Zero Page X
	EOR nn,X	5D	4+	3	XOR to A	Absolute X
	EOR nn,Y	59	4+	3	XOR to A	Absolute Y
I	INC nn	EE	6	3	Increment by one	Absolute
	INC n	E6	5	2	Increment by one	Zero Page
	INC n,X	F6	6	2	Increment by one	Zero Page X
	INC nn,X	FE	7	3	Increment by one	Absolute X
	INX	E8	2	1	Increment X by one	None
	INY	C8	2	1	Increment Y by one	None
J	JMP nn	4C	3	3	Jump to new location	Absolute
	JMP (nn)	6C	5	3	Jump to new location	Indirect
	JSR nn	20	6	3	Jump to subroutine	Absolute

	INSTRUCTION	OP	C	B	DESCRIPTION	ADDRESSING
L	LDA #n	A9	2	2	Load A	Immediate
	LDA nn	AD	4	3	Load A	Absolute
	LDA n	A5	3	2	Load A	Zero Page
	LDA (n,X)	A1	6	2	Load A	Ind X
	LDA (n),Y	B1	5+	2	Load A	Ind Y
	LDA n,X	B5	4	2	Load A	Zero Page X
	LDA nn,X	BD	4+	3	Load A	Absolute X
	LDA nn,Y	B9	4+	3	Load A	Absolute Y
	LDX #n	A2	2	2	Load X	Immediate
	LDX nn	AE	4	3	Load X	Absolute
	LDX n	A6	3	2	Load X	Zero Page
	LDX nn,Y	BE	4+	3	Load X	Absolute Y
	LDX n,Y	B6	4	2	Load X	Zero Page Y
	LDY #n	A0	2	2	Load Y	Immediate
	LDY nn	AC	4	3	Load Y	Absolute
	LDY n	A4	3	2	Load Y	Zero Page
	LDY n,X	B4	4	2	Load Y	Zero Page X
	LDY nn,X	BC	4+	3	Load Y	Absolute X
	LSR nn	4E	6	3	Logical shift right	Absolute
	LSR n	46	5	2	Logical shift right	Zero Page
	LSR A	4A	2	1	Logical shift right	Accumulator
	LSR n,X	56	6	2	Logical shift right	Zero Page X
	LSR nn,X	5E	7	3	Logical shift right	Absolute X
N	NOP	EA	2	1	No operation	None
O	ORA #n	09	2	2	OR to A	Immediate
	ORA nn	0D	4	3	OR to A	Absolute
	ORA n	05	3	2	OR to A	Zero Page
	ORA (n,X)	01	6	2	OR to A	Ind X
	ORA (n),Y	11	5+	2	OR to A	Ind Y
	ORA n,X	15	4	2	OR to A	Zero Page X
	ORA nn,X	1D	4+	3	OR to A	Absolute X
	ORA nn,Y	19	4+	3	OR to A	Absolute Y
P	PHA	48	3	1	Push A onto stack	None
	PHP	08	3	1	Push P onto stack	None
	PLA	68	4	1	Pull (pop) A from stack	None
	PLP	28	4	1	Pull (pop) P from stack	None
R	ROL nn	2E	6	3	Rotate left through carry	Absolute
	ROL n	26	5	2	Rotate left through carry	Zero Page
	ROL A	2A	2	1	Rotate left through carry	Accumulator
	ROL n,X	36	6	2	Rotate left through carry	Zero Page X
	ROL nn,X	3E	7	3	Rotate left through carry	Absolute X
	ROR nn	6E	6	3	Rotate right through carry	Absolute
	ROR n	66	5	2	Rotate right through carry	Zero Page
	ROR A	6A	2	1	Rotate right through carry	Accumulator
	ROR n,X	76	6	2	Rotate right through carry	Zero Page X
	ROR nn,X	7E	7	3	Rotate right through carry	Absolute X
	RTI	40	6	1	Return from interrupt	None
	RTS	60	6	1	Return from subroutine	None
S	SBC #n	E9	2	2	Subtract with borrow from A	Immediate
	SBC nn	ED	4	3	Subtract with borrow from A	Absolute
	SBC n	E5	3	2	Subtract with borrow from A	Zero Page
	SBC (n,X)	E1	6	2	Subtract with borrow from A	Ind X
	SBC (n),Y	F1	5+	2	Subtract with borrow from A	Ind Y
	SBC n,X	F5	4	2	Subtract with borrow from A	Zero Page X
	SBC nn,X	FD	4+	3	Subtract with borrow from A	Absolute X
	SBC nn,Y	F9	4+	3	Subtract with borrow from A	Absolute Y
	SEC	38	2	1	Set carry	None
	SED	F8	2	1	Set decimal mode	None
	SEI	78	2	1	Set IRQ disable	None
	STA nn	8D	4	3	Store A	Absolute
	STA n	85	3	2	Store A	Zero Page
	STA (n,X)	81	6	2	Store A	Ind X
	STA (n),Y	91	6	2	Store A	Ind Y
	STA n,X	95	4	2	Store A	Zero Page X
	STA nn,X	9D	5	3	Store A	Absolute X
	STA nn,Y	99	5	3	Store A	Absolute Y
	STX nn	8E	4	3	Store X	Absolute
	STX n	86	3	2	Store X	Zero Page
	STX n,Y	96	4	2	Store X	Zero Page Y
	STY nn	8C	4	3	Store Y	Absolute
	STY n	84	3	2	Store Y	Zero Page
	STY n,X	94	4	2	Store Y	Zero Page X
T	TAX	AA	2	1	Transfer A to X	None
	TAY	A8	2	1	Transfer A to Y	None
	TSX	BA	2	1	Transfer S to X	None
	TXA	8A	2	1	Transfer X to A	None
	TXS	9A	2	1	Transfer X to S	None
	TYA	98	2	1	Transfer Y to A	None

Instruction Notes

ADC	A+DATA+C→A
BRK	Ignore I flag, Set B=1 Push return address+1 Push P Jump to IRQ vector
JSR	Push return address-1 Jump absolute
RTI	Pop P, Pop PC
RTS	Pop PC, Increment PC
SBC	A-DATA-$\overline{C}$ →A

Shift Instructions



Added Cycle Time

A (+) in the (C) column for branch instructions means:
Add 0 if branch not taken.
Add 1 if taken within page.
Add 2 if taken across pages.

A (+) in the (C) column for other instructions means:
Add 1 if indexing across page boundary.

Assembler Symbols

.	Assembler directive
#	Immediate addressing
$	Hex number prefix
@	Octal number prefix
%	Binary number prefix
'	ASCII character prefix
()	Indirect addressing
;	In col 1 for comment

Intentionally Blank

DO NOT PLACE ON HOT SURFACES

WHILE PREPARED WITH EXTREME CARE, THERE ARE NO WARRANTIES EXPRESS OR IMPLIED AS TO MERCHANTABILITY OR FITNESS FOR PURPOSE.

MICRO CHART
AUTHOR:
JAMES D. LEWIS

#101B