# 6502

## Retrocomputer

**6502 Based Retrocomputer Emulation System**
**Copyright © 1976, 2021**
**Second Edition with Addendum**

Foreword
--------

Welcome to 1979. Jimmy Carter has just won the US presidency, and Rocky is showing in the cinema. It's a great year for NASA as they unveil the first space shuttle, the Enterprise, and the Viking craft land on Mars. A small computer company called 'Apple' has just been founded, and they are selling the Apple 1 Computer for $666.66

After six months of doing paper-rounds, and a gift from your favorite Aunt, you've finally been able to buy your very own computer: the one you've been reading about in magazines - a KIM-1.

You love the KIM-1, but you long for something a little better - so after six months you sell it, and after pleading with your parents, they help you buy an Apple-1. Now things really start to get interesting.

Welcome to this 6502-based hardware emulation system. It currently emulates two very basic computers systems: the KIM-1, and the Apple-1. Both were early computing pioneers, and used the 8-bit 6502 as their main CPU, with some memory and a few supporting chips. Neither had dedicated graphics hardware, and communicated either via LEDs or simple terminal input/output to a printer or display, and a keyboard.

Simple as they were, they helped create the home computer revolution. Hobbyists used them to write games, utilities, more games and this encouraged the development of better, faster, cheaper systems - and now you have one in your hand, capable of emulating its own ancestor.

**Note** These computers are NOT user-friendly. Think of this app as a retrocomputing puzzle in which you need to invest a little time to understand. The information is all out there.. somewhere! Enjoy :-)

                                                    -john

Table of Contents
-----------------

Foreword

1. Description of included systems

2. The KIM-1

3. The Apple-1

4. The MOS 6502

5. Additional Resources

# Chapter 1. Description of included systems
-----------------------------------------

Using a computer from this era might be a little shock-
ing for you. There is no operating system. There are no
graphics. There is a CPU, and some memory, and the rest
is up to you. This is very low-level computing. If you
have only used laptops or iPads, this way of thinking
may be quite alien, and you'll need to do a little read-
ing and research (see Chapter 5).

## The KIM-1
---------
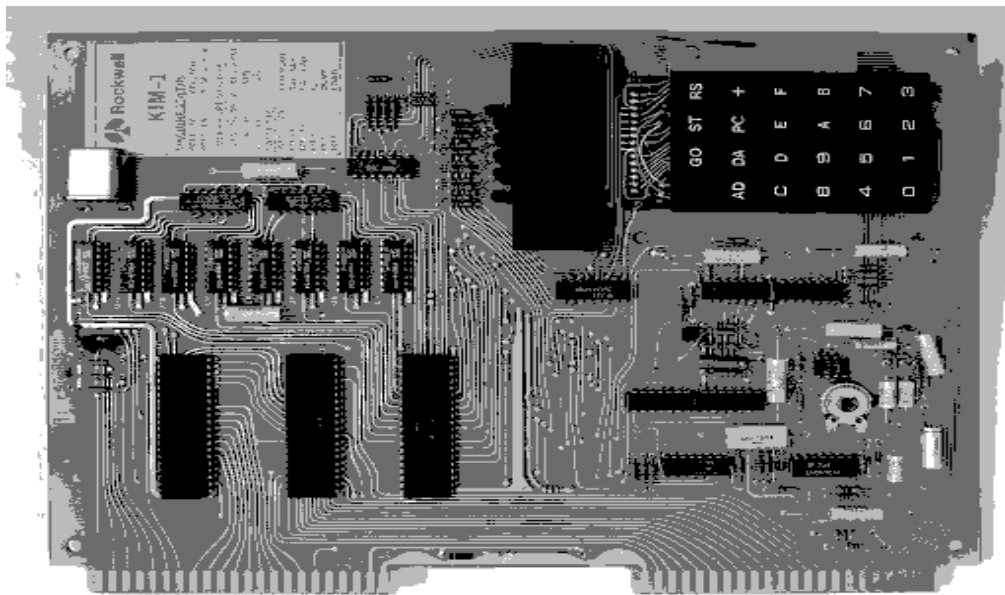


The KIM-1, short for Keyboard Input Monitor, is a small
6502-based single-board computer developed and produced
by MOS Technology, Inc. You had to provide your own pow-
er supply circuit before you could use it – and then you
needed to borrow a casssette recorder for storing pro-
grams. If you were lucky, you might be able to connect
it to a teletype terminal, or maybe get to use it with a
keyboard and screen (if you were rich).

The emulated KIM in this app has two modes – the origi-
nal hex keypad and LED display, and the console display.
From either, you can enter your own assembly code, or
select from some existing applications. You need to ap-
preciate that this is a very early computer, so set your
expectations accordingly!

# The Apple-1
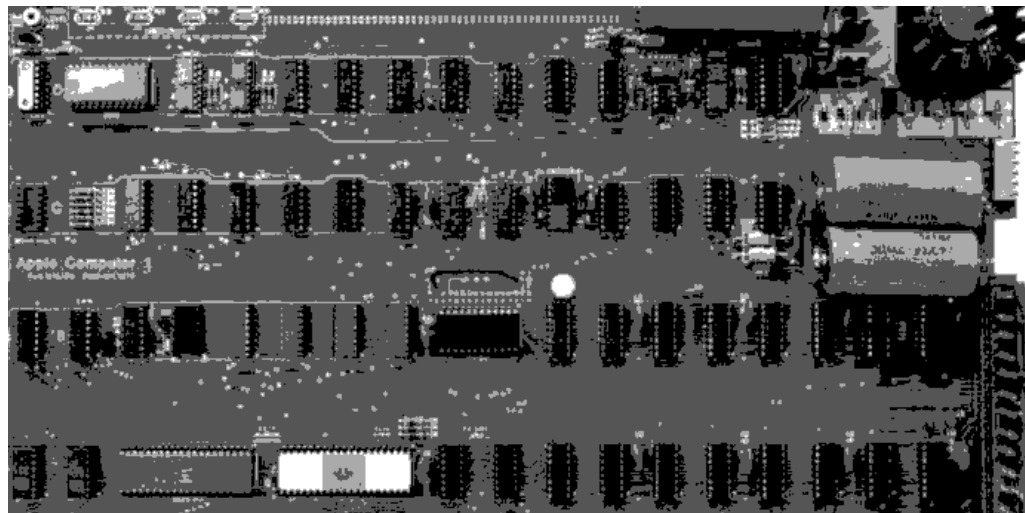


The Apple-1 was a ground-breaking computer from a small start-up called "Apple Computer Co." For your $666.66 you received a populated circuit board, and you needed to find the case, keyboard, power supply and a display.

As the Apple-1 is also 6502-based, it's possible to simulate it using the same CPU emulation used for the KIM-1. The addition of the 'Woz Mon' and BASIC means it can load and run classic games from Creative Computing magazine.

## Software

This emulator comes with several pieces of software for each device, to save you some typing.

You can of course type in your own programs. Before you spend time on this endeavor, please note:

a) There is currently no way to save your work.

b) Some applications depend on specific hardware features which may not be fully implemented by this emulator.

Chapter 2. The KIM-1
--------------------

The KIM-1 was sold as a trainer / technology demonstra-
tor single-board computer by MOS in 1976 to show off the
power of the 6502.

The KIM-1 hardware consists of a 6502 CPU, some memory
(just over 1Kb in a standard system), and two chips in
the 6502-family called the 6530 RIOT chips. These chips
provided support for input/output functions, some timer
features, and each had some ROM containing the KIM-1's
monitor program.

The display comprised six 7-segment displays, and input
was via a hex keypad with some extra functions. A cas-
sette interface was available to save and load programs
to tape.

The KIM-1 could also drive a serial port (slowly) to
connect to a terminal (VDU and keyboard).

Memory expansion kits were available (adding 4Kb of RAM
meant a version of BASIC could be used). Another kit was
available to display 16 lines of 32 upper-case charac-
ters on a TV, and take input from a keyboard.

The following page shows you the "memory map" of the
KIM-1, and it may not make a lot of sense right now.
That's ok. It does look cool though. Why not make a copy
and send it to your friends? They'll be impressed. All
it really shows you is how the memory of the KIM-1 is
organized, from the first address at 0000h all the way
up to 2000h and beyond. Highlights include the free mem-
ory at 0200h for your programs, the location of the KIM-
1 monitor ROM and the page reserved for the stack (100h
to 1FFh). Your virtual KIM-1 comes with all the expan-
sion memory you need (at no extra cost!) and a termi-
nal too, so you really are the envy of all your geek
friends.

## 2.1 KIM-1 Memory Map
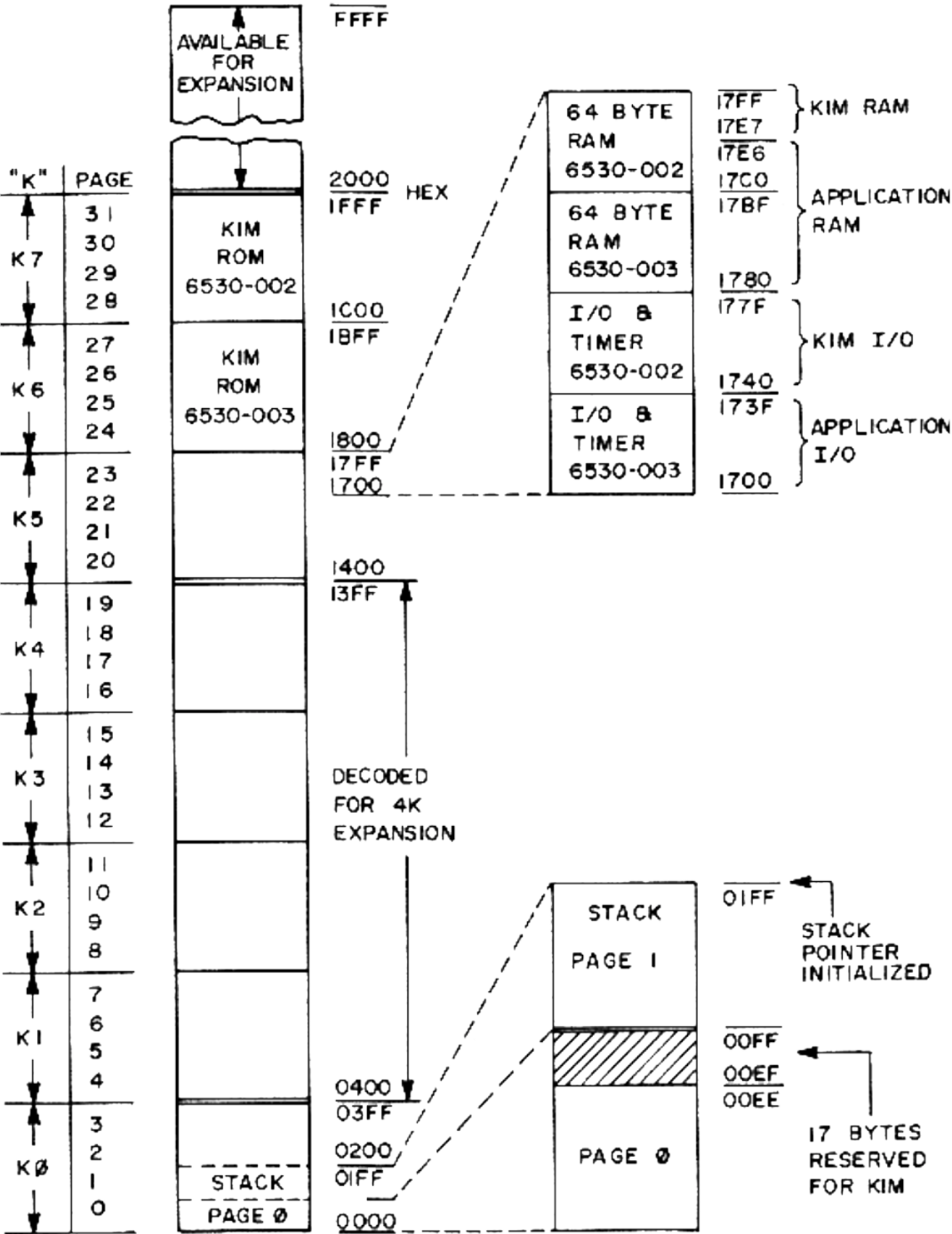
## 2.2 KIM-1 - Keypad and LED Mode
-------------------------------

In this mode, you enter instructions and code via the
keypad, and the six LEDs provide information. That's it.
That's all you get. However, this is still enough to try
out some applications or learn the basics of 6502 ma-
chine code programming.

The KIM-1 monitor program present in ROM (starting at
address $1f00) allows you to examine memory locations,
enter new values into memory, start executing a specific
address, and optionally step instruction-by-instruction
through code.

You can enter 6502 instructions manually from the key-
pad, or load software from the built-in catalog that em-
ulates having a cassette tape collection.

Why did the KIM-1 come with such basic input and output?
Cost! A dedicated terminal (the kind that looks like a
typewriter) cost a lot of money, and was both huge and
noisy. No sane kid's parents would let them have one.

The option of a TV or Monitor and keyboard was something
you probably couldn't even buy if you wanted to - build-
ing your own from a project in a magazine such as the
"TV Typewriter" by Don Lancaster was the only realistic
option for many users. (The fact the Apple-1 came with
the hardware to display text on a monitor as standard
was a big reason for its success.)

Even with only LEDs and keypads, hobbyists wrote simple
games. There was even a chess program that could play a
good match, and it used only the LEDs to tell you its
moves.

## 2.3 Operating Instructions
-------------------------

The KIM-1 has six 7-segment LED displays, and a keypad.
Once turned on, the keypad works as follows:

GO ..... Starts executing code at the current location.

ST ..... Stop any currently running code.

RS ..... Reset the system.

SST .... Toggle single-step mode.
         GO will stop after one instruction if active.

AD ..... Address mode: numbers entered are an address.

DA ..... Data mode: numbers entered will be stored in
         memory.

+ ...... Increment to the next address.

0 to F . Used to enter memory addresses or values, in
         hexidecmal (base 16).


These are enough commands, believe it or not, to en-
ter programs into the KIM-1 and start them. And you can
definitely do that. However, there's also a "LOAD" but-
ton at the bottom of the screen which will load some
software for you.

Also, after a RESET, the Lunar Lander game is already
loaded into memory, at location 0200. You should be able
to work out how to start it with the information above!

Note: If you find some programs in "paper tape for-
mat" for the KIM-1, you can paste them into the Load-
ing screen. Paper tape format is a string of hex digits
starting with a ";". No other format is currently sup-
ported.

## 2.4 KIM-1 – Console Mode
-----------------------

In Console Mode, the KIM-1 sends and receives input to a virtual screen and provides an on-screen keyboard. Use the shift key to access other important characters on the keyboard.

A real KIM-1 could connect to these devices over a relatively slow serial connection, and so the performance you see isn't particularly inacurate.

The KIM-1 monitor program allows for displaying and changing memory, and loading, and executing code. The big difference to the LED mode is that the programs can display text (upper-case only), rather than only displaying numbers of patterns on the LEDs. This makes for considerably more useful programs: The users of the KIM-1 were keen to stretch its capabilities, and so there were versions of BASIC, FORTH and other languages and utilities – most of which were used to create games.

## 2.5 Using the KIM-1 Monitor

When the KIM-1 starts connected to a terminal, you will
see:

        KIM
        0000 00

displayed. If you do not see this, press the Reset but-
ton. The numbers after KIM are an address, and the con-
tents of the memory at that address. All numbers are
displayed in hexadecimal.

To select a specific address, type in four numbers and
press SPACE. For example:

        1C00 <space>

and you will see

        1C00 85

This means the value 85, in hex, is currently at memory
location 1C00.

Press the RETURN key, and the next address will be dis-
played:

        <return>
        1C01 F3

To change the contents of memory, use a period. For ex-
ample:

        0000 <space>
        42.

You will see:

        0001 00 (or maybe another value)

displayed.

This will select memory address 0000, and then poke the
value of 42 into that memory location. The monitor will
then automatically move to location 0001, which makes it
easy to enter new values one after the other.

Let's check that 42 was put in location 0000:

        0000 <space>

This will (hopefully) display:

        0000 42

There aren't many options built into the KIM-1 monitor.
In fact, the only other commands are:

G .. Go to the current location and start executing code
L .. Load a program from cassette tape (not supported)
Q .. Dump some memory to the screen (not super useful)

For now, you should use the LOAD button at the top of
the screen to load some of the sample software built
into the emulation, and enjoy some state-of-the-art
games (circa. 1976).

## 2.6 KIM-1 Software
------------------

LUNAR LANDER
by Jim Butterfield

This program starts at 0200.  When started, you will
find yourself at 4500 feet and falling.  The thrust on
your machine is set to low; so you will pick up speed
due to the force of gravity.

You can look at your fuel at any time by pressing the
"F" button.  Your fuel (initially 800 pounds) will be
shown in the first four digits of the KIM display.

The first two digits of the KIM display always show
your rate of ascent of descent.  "A" restores altitude.

Set your thrust by pressing buttons 1 through 9.
Warning: button 0 turns you motor off, and it will not
reignite!  A thrust of 1, minimum, burns very little
fuel; but gravity will be pulling your craft down faster
and faster.  A thrust of 9, maximum, overcomes gravity
and reduces your rate of descent very sharply  A thrust
of 5 exactly counterbalances gravity; you will continue
to descend (or ascend) at a constant rate.  If you run
out of fuel, your thrust controls will become inopera-
tive.

Suggestions for a safe flight:
  [1]    Conserve fuel at the beginning by pressing 1.
         You begin to pick up speed downwards.
  [2]    When your rate of descent gets up to the 90's,
         you're falling fast enough.  Press 5 to steady
         the rate.
  [3]    When your altitude reaches about 1500 feet,
         you'll need to slow down.  Press 9 and slow down
         fast.
  [4]    When your rate of descent has dropped to 15 or
         20, steady the craft by pressing 5 or 6.

Now you're on your own.

ADDITION
by Jim Butterfield

HERE'S A HANDY LITTLE ADDING MACHINE PROGRAM.  KIM BE-
COMES A SIX DIGIT ADDER.  "GO" CLEARS THE TOTAL SO YOU
CAN START OVER.  THEN ENTER A NUMBER AND HIT THE PLUS
KEY TO ADD IT TO THE PREVIOUS TOTAL.  IF YOU MAKE A MIS-
TAKE IN ENTERING A NUMBER, JUST HIT THE "0" KEY SEVER-
AL TIMES AND ROLL THE BAD NUMBER OUT BEFORE ENTERING THE
CORRECTION.  NO OVERFLOW INDICATOR, AND NO SUBTRACTION
OR MULTIPLICATION – MAYBE YOU WOULD LIKE TO TRY YOUR
HAND AT ADDING THESE. THE PROGRAM IS FULLY RELOCATABLE,
BUT IS LOADED AT 0200. ENTER 0200 AND PRESS GO TO START.


HI-LO
by Jim Butterfield

AN EASY GAME FOR ONE OR MORE PLAYERS.  KIM CHOOSES A SE-
CRET NUMBER FROM 01 TO 98.  AT THE START, THE FIRST FOUR
DIGITS SHOW THE HIGH AND LOW BOUNDS OF THE NUMBER – 99
HIGH AND 00 LOW.  AS GUESSES ARE ENTERED – ENTER THE
GUESS AND PRESS A FOR ATTEMPT – THE BOUNDS CHANGE AS
YOU ARE NARROWING DOWN THE POSSIBILITIES.  FOR EXAMPLE,
GUESS 32 AND THE DISPLAY MIGHT CHANGE TO 32 00, MEAN-
ING THAT THE COMPUTER'S SECRET NUMBER IS BETWEEN THESE
VALUES.  AFTER EACH LEGAL GUESS, THE COMPUTER SHOWS THE
NUMBER OF ATTEMPTS MADE SO FAR.

ONE PLAYER GAME: TRY TO GET THE MYSTERY NUMBER IN SIX
ATTEMPTS.

MULTI PLAYER GAME: EACH PLAYER TRIES TO AVOID GUESSING
THE MYSTERY NUMBER – THE CORRECT GUESSER LOSES AND IS
"OUT".

THE GAME IS LOADED AT 0200. ENTER 0200 AND PRESS GO TO
START.

TIMER
by Jim Butterfield

TIMER turns KIM into a digital stopwatch showing up to
99 minutes and 59.99 seconds. It is designed to be ac-
curate to 50 microseconds per second. The interval tim-
er is used to count 9984 cycles and the instructions be-
tween the time out and the reset of the timer make up
the other 16 cycles in .01 seconds. The keyboard is used
to control the routine as follows: Stop (0), Go (1), Re-
turn to KIM {4) , Reset (2).

## 2.7 KIM-1 Console mode Software

The following programs are available in Console mode.

### BASIC

An implementation of BASIC for the KIM-1. Once loaded, enter 4068 <space> G to start it.

Press enter at the memory and width questions, and Y and enter for SIN-COS-TAN.

### JMON

A powerful monitor program, which can display edit memory and even disasssemble code. Once loaded, enter 2000 <space> G to start. Enter a ? for a list of options.

### FORTH

A version of FIG-FORTH. Currently not very stable. Once loaded, enter 2300 <space> G to start. Try entering 1 2 3 4 + . to get a feel for Reverse Polish Notation.

### DISASSEMBLER

A useful disassembler. Once loaded, you can create a listing of 6502 instructions by entering 2000 <space> G. Note that the disassembler will start disassembling at the address contained in 0000h and 0001h (low byte first).

### TICTACTOE

This is a game written in BASIC. Press Load, select Tic-TacToe and then press the Warm Reset button. Now start the game by typing RUN.

Chapter 3. The Apple-1
----------------------


3.1 Apple-1 Specifications
--------------------------

The Apple-1 was a 6502 based computer, available in kit
form for $666.66. It first went on sale in July 1976.

It came with a 4Kb of RAM, and a 256 byte ROM containing
the "WozMon" monitor program. A 6820 was used for input/
output, and other hardware implemented memory refresh
and the built-in terminal display hardware for a 40 by
24 character display.

BASIC was provided on cassette tape, which also required
the used of an cassette interface card (available for an
extra $80).

Compared to other systems of the time, the Apple-1's
price and built-in video made it an attractive system
for hobbyists.

```
3.2 WozMon
----------


The Apple-1 starts with the WozMon running: a monitor
program for displaying and editing memory. The \ charac-
ter is the WozMon prompt.

Enter an address to see the contents of memory, e.g.

    300 <enter>
    0301: 00


This display two addresses separated by a period to see
the contents of a block of memory, e.g.

    300.310 <enter>

    0300: 10 00 00 00 00 00 00 00
    0308: 00 00 00 00 00 00 00 00
    0310: 00


Use a period followed by an address to continuing dis-
playing memory, e.g.

    .320 <enter>

    00 00 00 00 00 00 00 00
    0318: 00 00 00 00 00 00 00 00
    0320: 00


To change the memory contents, use a : symbol, e.g.

    300: 42 <enter>

    0300: 10


The '10' is what the memory used to contain. Check the
new value is correct with 300. <enter>

    0300: 42
```

WozMon (Continued..)
--------------------

To execute code, use the R command, e.g.

    FFDC R

This will jump to code at FFDC which will display a
byte. It will then lock up the emulator, so press Reset.

The WozMon is a useful tool for entering and running
6502 assembly code, although it is not an assembler: you
will need to convert the 6502 instructions into their
numberic codes and enter them manually (see section
4.5). This conversion process is something that gets old
very quickly. One option is to use an assembler, such as
the web-based ASM80.COM tool to convert your instruc-
tions into machine code.

The Apple-1 also has all the memory you could need, but
the memory map is a little different. The first 256 bytes
of memory are "zero page", as in all 6502 systems, and
the second 256 are used for the stack (again, as all
6502 systems do). After that, things diverge. However,
as long as you start your programs at $300 (rather than
$200 on the KIM-1) you should be fine). Plenty more in-
formation on the Apple-1 (included scans of the original
manual) are available in all good bookstores (I mean, on
the Internet).

## 3.3 Apple-1 Software
--------------------

### Star Trek
---------

A game written in BASIC. Once loaded, press Warm Reset
and enter RUN.

### Star Trek VB
------------

A more advanced Star Trek game, also in BASIC. Once
loaded, press Warm Reset and enter RUN.

### 15 Puzzle
---------

A puzzle game written in machine code. Start by entering
300 R <enter> once loaded.

### Shut the Box
------------

A puzzle game written in machine code. Start by entering
300 R <enter> once loaded.

### KINGDOM
-------

A game written in BASIC. Once loaded, press Warm Reset
and enter RUN.

### SLOTS
-----

A game written in BASIC. Once loaded, press Warm Reset
and enter RUN.

# Chapter 4. The MOS 6502

## 4.1 Introduction



The 6502 CPU was launched as an industrial control chip
in 1977, and its main claim to fame was its low price
compared to rival CPUs. It was designed by a team of
ex-Motorola engineers who had created a CPU called the
6800. Disappointed that the 6800 was expensive and
therefore of limited appeal, many of the 6800 team moved
to rival company MOS, where they designed a CPU called
the 6501 (which was pin compatible with the 6800, cheap-
er, simpler, faster and therefore the source of a legal
challenge). As a result, they then built the 6502.

The relatively low price of the CPU made it the dar-
ling of electronic hobbyists who started building com-
puters based around it: something that the original de-
signers (a team led by Chuck Peddle) had not envisaged.
Alongside the 6502, MOS created a single-board training
system called the KIM-1. The KIM-1 was designed for en-
gineers to try out the CPU for themselves, and then may-
be build it into new products. It too was snapped up by
hobbyists, and an entire ecosystem of magazines, soft-
ware and peripherals grew up around it.

MOS was soon purchased by Commodore, who continued to
produce and license the design. As well as Commodore,
other companies started basing their own computer sys-
tems around the 6502 including Atari, Acorn, Apple,
Oric, and Nintendo. There is no doubt that, perhaps in-
advertently, the 6502 helped kick off the home computer
revolution of the 1980s.

## 4.2 CPU Concepts
----------------

Computers are still based on a CPU - a Central Processing Unit - an electronic circuit designed to access memory and perform calculations and make decisions. They are fundamentally simple machines, but the ability to perform these limited actions extremely quickly allows sophisticated software to be created - everything from games to graphical user interfaces and even emulation systems like this one.

Writing code for the CPU was often the only option in the early days of computing, until languages such as BASIC were developed to make things a little easier. Using the CPU's Assembly Language is both simple (there are only a limited set of instructions and what they can do) and complicated (for exactly the same reasons). Remember that BASIC itself is just a program, written in assembly language, that converts commands such as PRINT into a series of other assembly language instructions.

To the CPU, a program consists of assembly language instructions stored in memory and read and executed one instruction at a time. If you were to look at a memory location at random, all you would see is a number - and you couldn't tell if the number represented a number (e.g. 10 or 0Ah) or an assembly language instruction (e.g. "ASL A" or "shift the bits in the accumulator on position to the left"). It's up to you, the programmer, to put the numbers in the right order and tell the CPU where to start executing code.

## 4.3 6502 Architecture
---------------------

The 6502 CPU is called an "8-bit CPU", because it has an 8-bit data bus, and a 16-bit address bus. With 8 bits of data, a register or memory address can store a number between 0 and 255 (or -128 to 127 if you choose to use two's complement arithmetic). With 16 bits for addresses, the CPU can access up to 64Kb of RAM (although early systems rarely came with 64Kb as standard, simply because memory was very expensive.

Compared to other processors of the time such as the Intel 8080, the 6502 is relatively simple with a limited number of internal registers. The first 256 bytes of memory are known as "zero page" memory, and the CPU has some addressing modes that make accessing this memory particularly fast and convenient.

The second 256 bytes are reserved for the system stack, which is where return addresses are stored when calling subroutines. The SP register is an 8-bit register that references the memory in this second page. e.g. if the SP contains FFh, then the actual memory address it is accessing is 01FFh.

Any I/O devices also appear in the memory map – there are no separate I/O instructions. For the KIM-1, for example, you can access the RIOT chip's timer by reading and writing to memory address 17FFh.

The last locations in the memory map store vectors – addresses which are called when the system is reset or given an interrupt signal.

## 4.4 Registers
-------------

The 6502 has the following registers:

The Accumulator is an 8-bit register used for arithmetic and logic operations.

X is an 8-bit register usually used in different types of index addressing modes.

Y is an 8-bit register usually used in different types of index addressing modes.

PC is the Program counter and is a 16-bit register.

SP is the Stack pointer and is an 8-bit register, and is always added to 100h to store the current location of the stack.

SR is the Status Register and contains the following flags:

Negative (N) ....... set if the most significant bit of the result is set.

Overflow (V) ........ set if an math operation overflows the 8 bits in the accumulator.

Break command (B) .. set if interrupt occurs.

Decimal mode (D) ... set if the processor is in decimal mode.

IRQ disable (I) .... set if the IRQ interrupt is disabled.

Zero (Z) ........... set if the result is zero.

Carry (C) .......... set if there was a carry from or borrow to during last result calculation.

# 4.4 6502 Opcodes

This table lists the instructions understood by the
6502. From it, you can determine the numeric value for
each instruction or "opcode". For example, TXA would
have the value 8Ah (h means "in hex"), and BNE would
have the value D0h.

Learning how to program in 6502 requires an understand-
ing of the instructions that available to you. Memorize
this chart if you can :-)

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

# 4.5 Writing a 6502 program
--------------------------

Let's use the table of opcodes to write a simple 6502 program to add two numbers, and multiply the answer by two.

Here's the intructions that will achieve this:

```
                // Start the code at address 0300h
LDA #$8         // Load 8 into the accumulator register
CLC             // Clear the carry flag before addition
ADC #$1         // Add 1 to the value in the accumulator
ASL             // Multiply accumulator by 2
STA $400        // Store the answer at address 0400h
BRK             // Stop
```

**Notes**

1. Hex numbers are preceded by $

2. The # means "this actual value, not the memory address". If we left out the #, then the accumulator would be loaded with the contents of memory address 0008h, not the value 8.

3. The carry flag needs cleared before every addition, or the answer will be incorrect. CLC does this for us.

4. Addition or substraction updates the contents of the accumulator i.e. it will be updated to contain the result of the operation.

4. ASL shifts every bit in the accumulator to the left. If you remember your binary arithmetic, this is the same as multiplying by two.

How do we enter this program into our computer?

First we need to convert every instruction, and the data that goes with it, into numeric codes using the opcode table.

Here is our program, this time with the opcodes (found
by looking up the instructions in the table):

```
LDA #$8        // A9 08
CLC            // 18
ADC #$1        // 69 01
ASL            // 0A
STA $400       // 8D 00 04
BRK            // 00
```
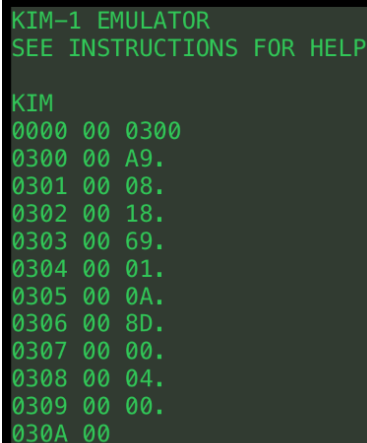
Now we have the opcodes, we can enter them into the com-
puter. Start the KIM-1 in Console Mode, and enter

        0300 <space>

to set the current address to 300h. Now enter the fol-
lowing numbers, pressing the period after each one:

        A9. 08. 18. 69. 01. 0A. 8D. 00. 04. 00.

Like this:

```
KIM-1 EMULATOR
SEE INSTRUCTIONS FOR HELP

KIM
0000 00 0300
0300 00 A9.
0301 00 08.
0302 00 18.
0303 00 69.
0304 00 01.
0305 00 0A.
0306 00 8D.
0307 00 00.
0308 00 04.
0309 00 00.
030A 00
```

Now return to the start of the program, by entering
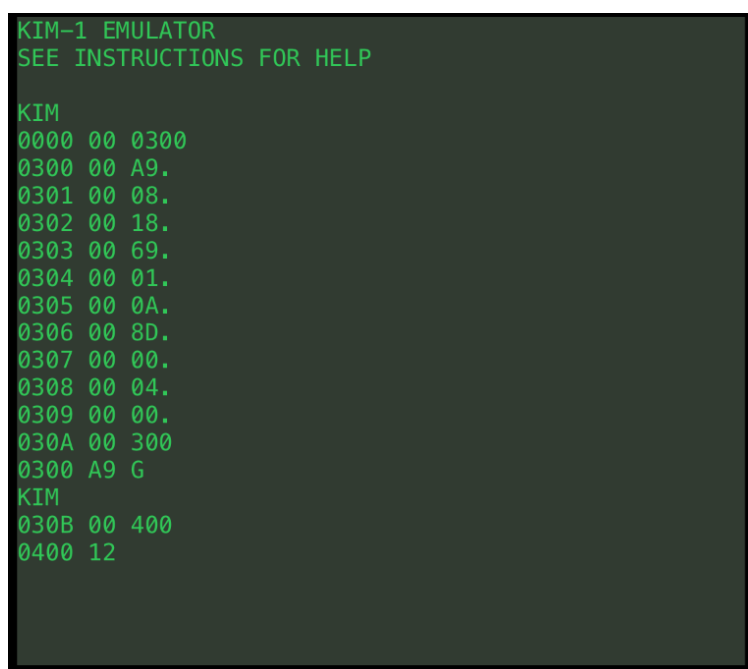
        0300 <space>

You can now start your program by entering G.

The program will run in a blink of the eye. You can now examine memory address 0400h to see what is there, by entering:

    0400 <space>

You should see something like this:

```
KIM-1 EMULATOR
SEE INSTRUCTIONS FOR HELP

KIM
0000 00 0300
0300 00 A9.
0301 00 08.
0302 00 18.
0303 00 69.
0304 00 01.
0305 00 0A.
0306 00 8D.
0307 00 00.
0308 00 04.
0309 00 00.
030A 00 300
0300 A9 G
KIM
030B 00 400
0400 12
```

The answer is 12. Wait a second: 8 + 1 multiplied by 2 is 12? Shouldn't it be 18? Did we make a mistake?

No, we got it right – remember the answer is 12 when US-ING HEXADECIMAL. 12 in hexadecimal is 18 in decimal.

Your first 6502 assembly language program worked!

Other instructions can perform operations such as test-ing the value of the accumulator and deciding to jump to a specific address – that is how you create loops and IF/THEN type conditional branches.

# Chapter 5. Additional Resources
--------------------------------

Everything you could possibly need to learn about pro-
gramming the 6502 is available on the internet. Classic
books are only a web search away, and can be downloaded
as PDFs into your device's Books app for easy reference.

Here are a list of titles to search for:

## KIM-1 Resources
--------------

    KIM-1 User Manual

    The First Book of KIM
        by Jim Butterfield

## Apple-1 Resources
----------------

    Apple-1 Operation Manual

    Assembly Lines: The Complete Book
    by Roger Wagner (in print and available to purchase)

## 6502 Resources
-------------

    Programming the 6502
    by Rodnay Zaks

    6502 Assembly Language programming
    by Lance A Leventhal


It is also possible to buy reproduction kits of the KIM-
1 and Apple-1 from other hobbyists, to recreate the
physical experience of owning a classic system. Search
for the PAL-1 on Tindie, or the Kim Uno, or the RC6502.

This page intentionally left blank
for no real reason.

Addendum
--------

Your virtual 6502 system now comes with a built-in 6502
assembler, at <u>no extra cost</u>!

Tap the ASM button at the top right of the screen, and
you will open a new source code file. There is a short
program there for you already.

Tap ASSEMBLE at the bottom of the screen, and the Source
Code will be assembled, and stored in memory. Now you
can tap the KIM-1 or APPLE 1 buttons under the code dis-
play, and the code will be copied into the memory of the
target system. (Tapping the KIM-1 or APPLE 1 buttons at
the top of the display will switch to that system but
also reset memory, as per usual).

Step by Step
------------

1. Tap ASM
2. Tap ASSEMBLE
3. Tap KIM-1 Console (the second KIM from the left)
4. Enter 200 and press SPACE
5. Enter G

You will see the digits 0 to 9 appear, just as the pro-
gram was written to do.

The characters are written to the screen by calling the
routine at $1EA0 in the KIM-1 ROM. There are a few other
ROM routines available see the next page for a list.

If you are writing code for the APPLE 1, then there are
different ROM routines available. Note that the APPLE 1
prefers code to be stored at $300, rather than $200.

## KIM-1 ROM Routines
------------------

These routines can be called with JSR on the KIM-1 in
both console and LED/hexpad modes.

```
$1EFE          Check for key being pressed
$1F6A          Get the value of key being pressed
$1F1F          Show the contents of memory F9, FA, and
               FB on the LEDs
$1E5A          Read a character entered at the console
               into the A registers
$1E3B          Print the contents of A as a hex number
$1E1E          Print the contents of memory FB and FA
$1EA0          Print the contents of register A as an
               ASCII character
$1E9E          Print a Space character
$1E2F          Print a new line
```

## APPLE 1 ROM Routines
-------------------

The APPLE 1 ROM monitor is tiny, but does contain some
code you can call:

```
$d010          Read a key from the console
$d012          Write a character to the console
```

```
Annotated KIM-1 Sample Program
------------------------------

        ; Sample program for KIM-1
        ; run this from the console
        ; and it will count from 0 to 9


        org $200      ; The start address for the code

print   equ $1ea0     ; ROM routine for displaying A
newline equ $1e2f     ; ROM routine for a new line

        jsr newline   ; Start by taking a new line
        lda #48       ; Load the value 48 (ASCII 0)
loop:   pha           ; Push A to the stack to keep it
        jsr print     ; Call the routine to print A
        jsr newline   ; Take a new line
        pla           ; Pop A back, as PRINT changed it
        adc #1        ; Add 1 to A
        cmp #58       ; If not equal to 58 ('9') then
        bne loop      ; jump back to loop
        brk           ; Stop the program.
```

The ROM routines are very useful, but they will often
change the value in any registers so be careful to keep
safe of any values you need before calling them.