Project: **My-KIM – A Replica of the KIM-1 Computer**

Title: **Hardware / Software Documentation**

	Name	Date
Author:	Manfred Langemann	01.04.2021

Table of Content

Table of Content

1 Summary

My-KIM represents a replica of the KIM-1 computer, developed in 1975 as a One-Board computer to become familiar with the MOS technology and their offered 6502 micro-controller family, see Figure 1-1. I bought this board in mid 1976 and learned how to program these controllers. Moreover I developed a dynamic memory expansion board with 16 KByte (at these times I thought that I would never get this memory full), added a teletype board and a monitor interface to realise a stand-alone computer with keyboard and TV monitor. On top of it I bought the Microsoft BASIC interpreter software for KIM-1, which was just released by the authors Bill Gates, Paul Allen and Monte Davidoff, representing their first commercial project.

Based on this I was the first student at the university, who wrote his thesis on his own computer – lucky guy. In the late 90th I disposed the entire computer, including the still functioning KIM-1 board at a recycling center – at that time I must have been very sick : poor guy, because today these historical boards are offered in the internet for more than 1000s of dollars.

Because winter time, corona with social restrictions and retirement being fell all on the same day of the week, I decided to rebuild this computer with mostly the original parts. The result of this work is described in the next chapters. To distinguish it from KIM-1, I named my developed computer “My-KIM”, which offers the following features:

- A complete 6502 Integrated Development Environment (IDE) for writing assembler source code, including editor, assembler, debugger and EEPROM programmer, see Figure 1-3 and Figure 1-4
- The hardware comes with a 6502 micro-controller, 8 KByte EEPROM, 32 KByte RAM and an I/O interface based on an Intel 8255 PPI (Programmable Peripheral Interface), offering three 8-bit ports. The audio tape interface is realised by an NE565 PLL and the RS232 serial interface is based on a MAX232 IC

The built My-KIM computer, see Figure 1-2, and the supporting IDE can be operated in four different modes:

Stand-alone Keyboard / Display Mode

In this mode the user can enter the source code in hexa-decimal values on the My-KIM keyboard and the results are shown on six 7-segment display. The user written code can be saved/loaded to/from a smartphone (headset input/output). User written programs can also be executed in the so-called Single Step mode, i.e. one instruction after the other.

Integrated Development Environment Mode

In this mode the user can write assembler code for the 6502 processor, including editing, assembling and debugging. The code can be flashed to an EEPROM or to a file, which can be loaded into the RAM of the My-KIM computer, see Figure 1-3.

Teletype Mode

In this mode My-KIM can be operated in the IDE via the Teletype mode, realised as a serial RS232 interface to a PC. My-KIM is then operated from an edit control window within the IDE. User written source codes can be saved/loaded on/from the PC or as well on a smartphone, see Figure 1-5.

BASIC Mode

In this IDE mode My-KIM runs the Microsoft BASIC interpreter V1.1 from 1977. The BASIC interpreter is loaded and operated from an edit control window within the IDE. User written BASIC programs can be saved/loaded on/from the PC in different files, see Figure 1-6.

Summary

Figure 1-1: Original KIM-1 Board, the Size is roughly a DIN-A4 Page

Figure 1-2: My-KIM, the Replica of KIM-1

Summary

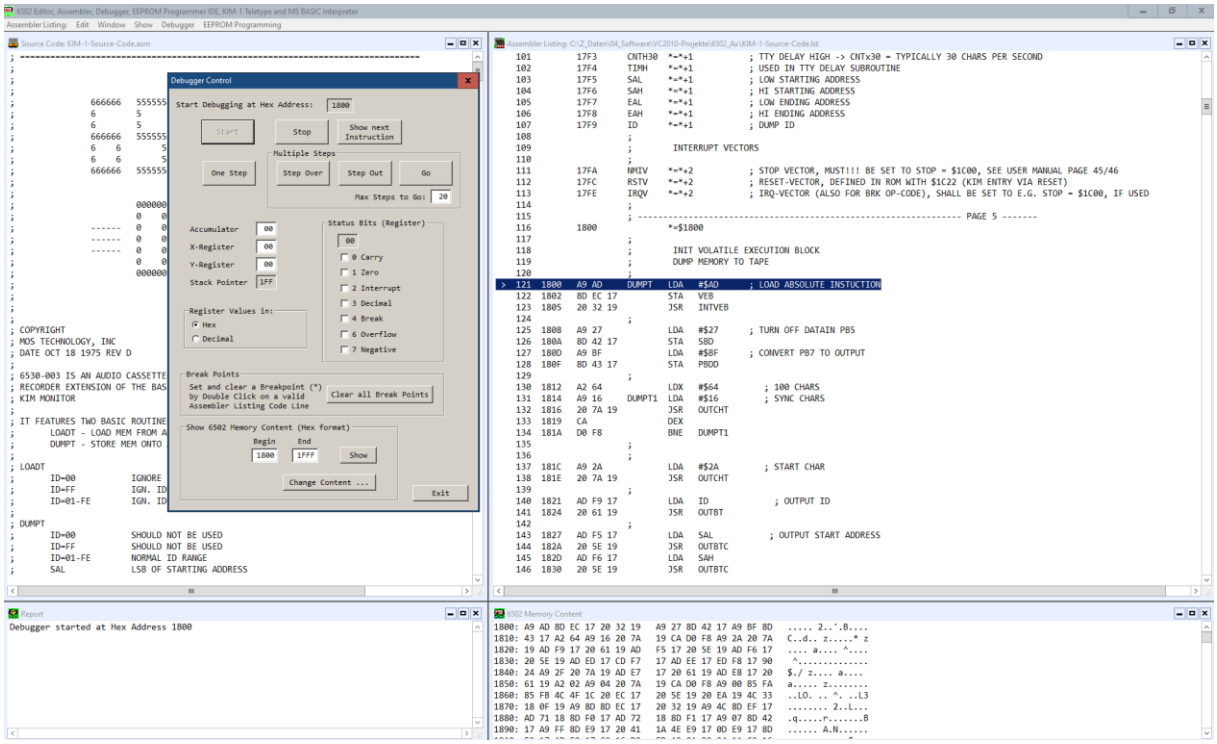


Figure 1-3: IDE showing a Debugging Session with the Original KIM-1 ROM Software

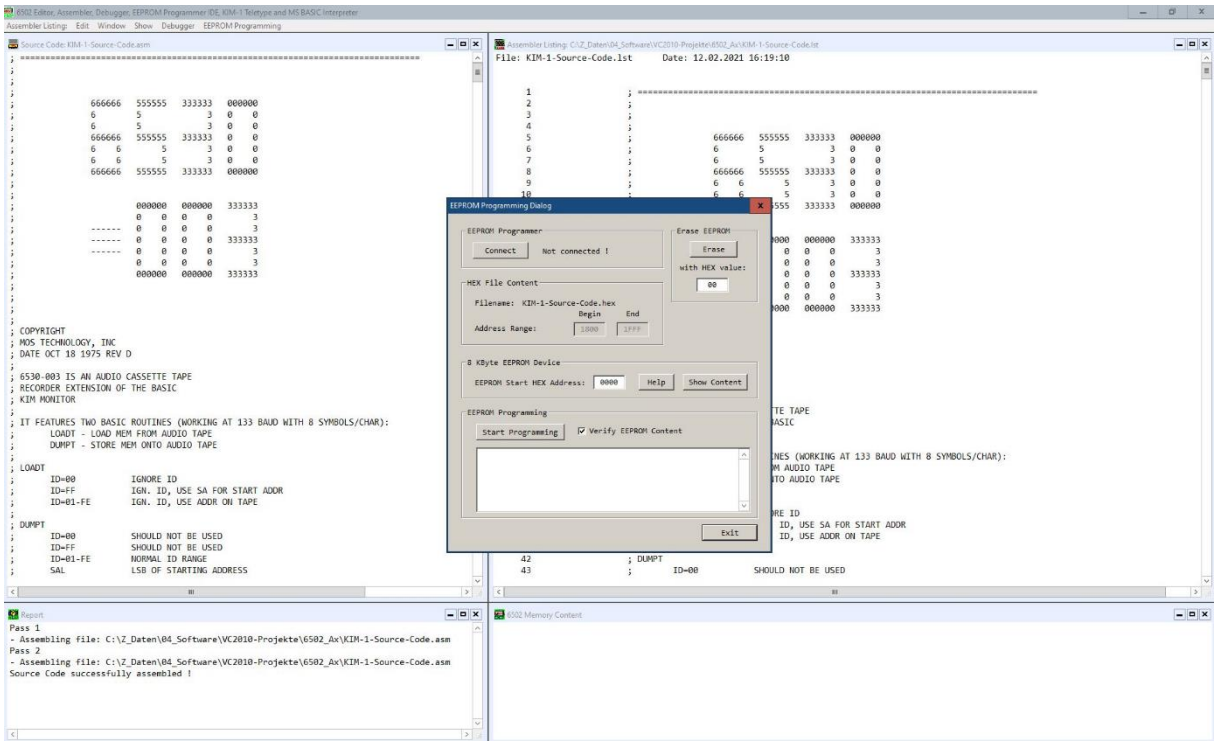


Figure 1-4: IDE showing an EEPROM Programming Session

Summary

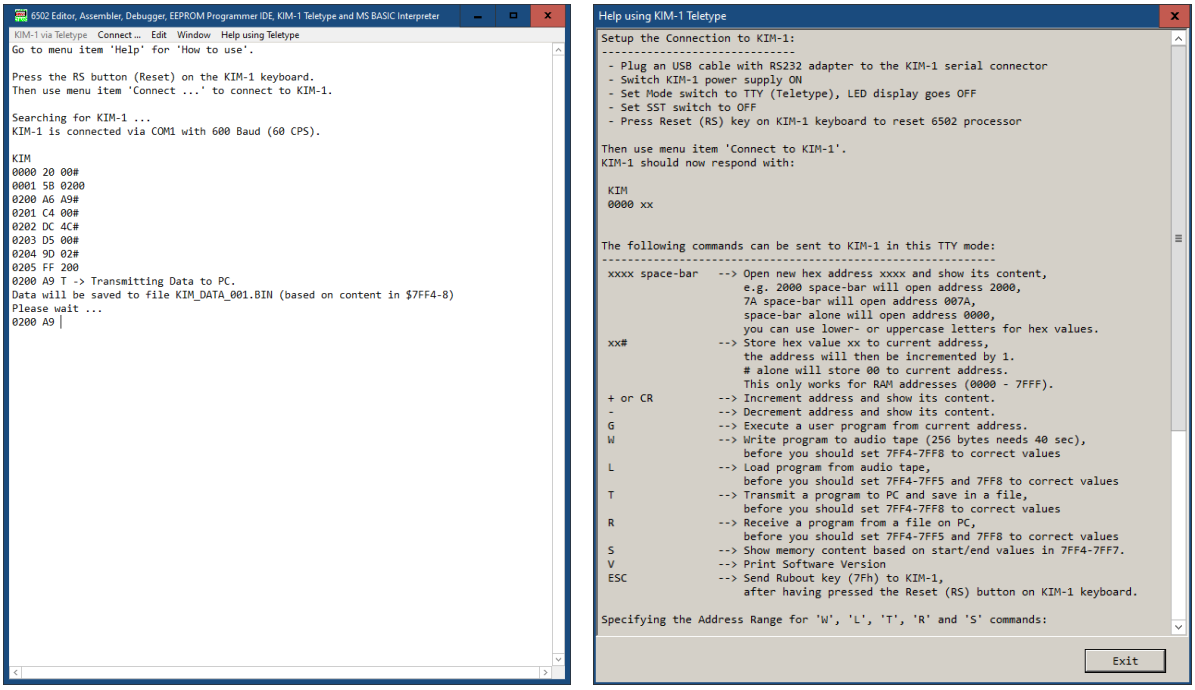


Figure 1-5: My-KIM operated in the Teletype Mode

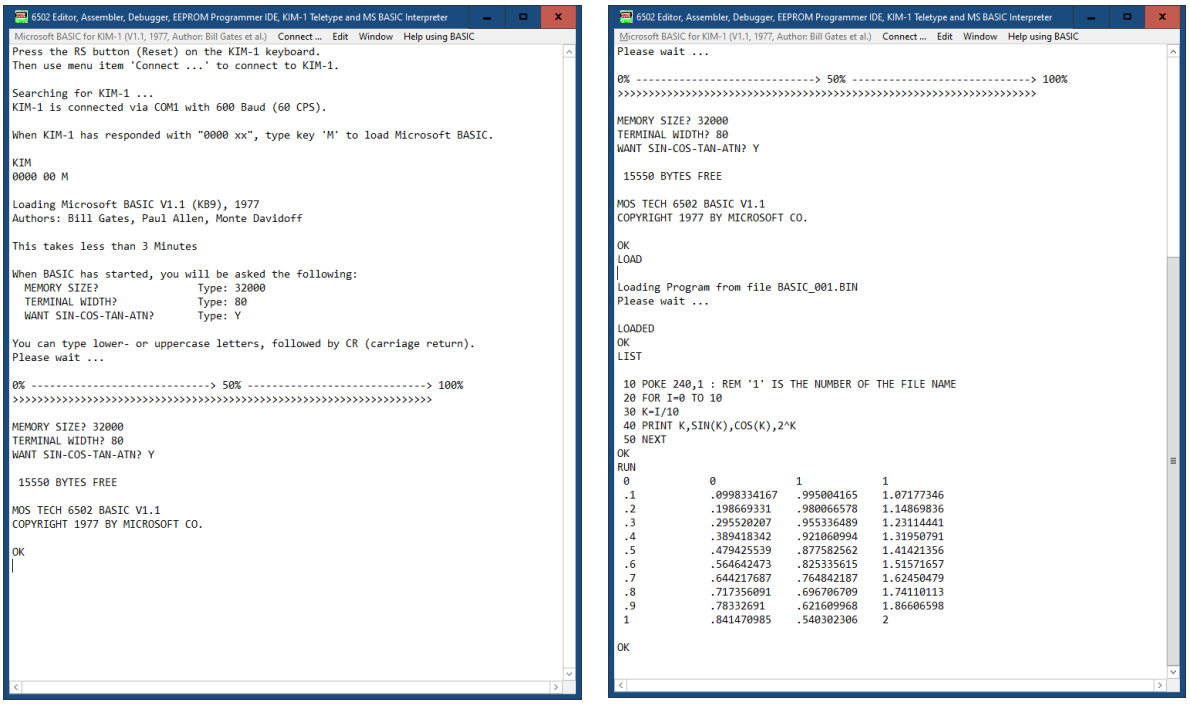


Figure 1-6: My-KIM operated in the Microsoft BASIC Mode

2 Hardware

2.1 Overview

Although I tried to mostly follow the design of KIM-1, I made some improvements to get a more flexible system regarding hardware and software. As shown in Table 2-1, KIM-1 only offers a RAM capacity of 1 KByte based on 8 x 6102 ICs, which was very limiting, in particular when the Microsoft BASIC interpreter shall be used, which already eats more than 8 KByte of memory. It was therefore decided to equip My-KIM with a very cheap single 32 KByte RAM of type 62256. To have some flexibility in adding additional software features in the operating system, an 8 KByte EEPROM was selected.

The two KIM-1 I/O ICs of type 6530 were specifically customised by MOS for the dedicated requirements of KIM-1 and are today no longer available. It was decided to replace these ICs by the historical Intel 8255 Programmable Peripheral Interface (PPI) IC, still available as used parts in the internet. The advantage of this IC is, that it is directly compatible with the control line architecture of the 6502 processor. However, this device has also disadvantages, which are:

- Only a complete 8-bit port can be set to either output or input, not the individual port lines
- Only the port C output lines can individually be set to 0 or 1 by the so-called BSR feature
- When changing the 8255 configuration (Control Word), all output ports will be reset to 0

These disadvantages have been compensated by suitable software adaptations.

Table 2-1: Hardware Differences between KIM-1 and My-KIM

	KIM-1	**My-KIM**
RAM	1 KByte, 8 x 6102	32 KByte, 1 x 62256
ROM	2 KByte (in 6530-002/3)	8 KByte (28C64 EEPROM)
I/O Systems	6530-002, 6530-003 with 4 x 8-Bit Ports	8255 PPI 3 x 8-Bit Ports
Audio Tape	Based on NE 565 PLL	Based on NE 565 PLL
Teletype	20 mA Feedback Loop	RS232 UART offering 600 Baud

Based on these design changes, the functional block diagram of My-KIM looks like as shown in Figure 2-1. It took a while to develop the correct control interface scheme of the 6502 processor with the RAM, EEPROM and 8255 ICs, because I didn't wanted the unpleasant case, having everything built up but, but nothing works. The finally chosen memory and I/O control scheme is shown in Figure 2-2.

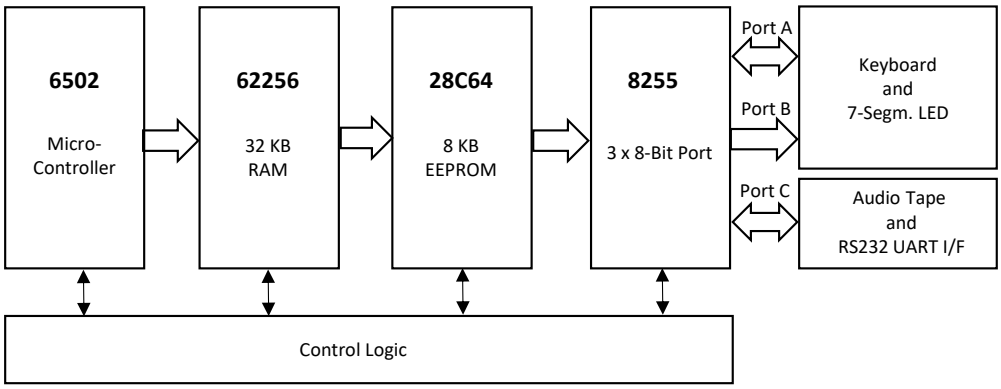
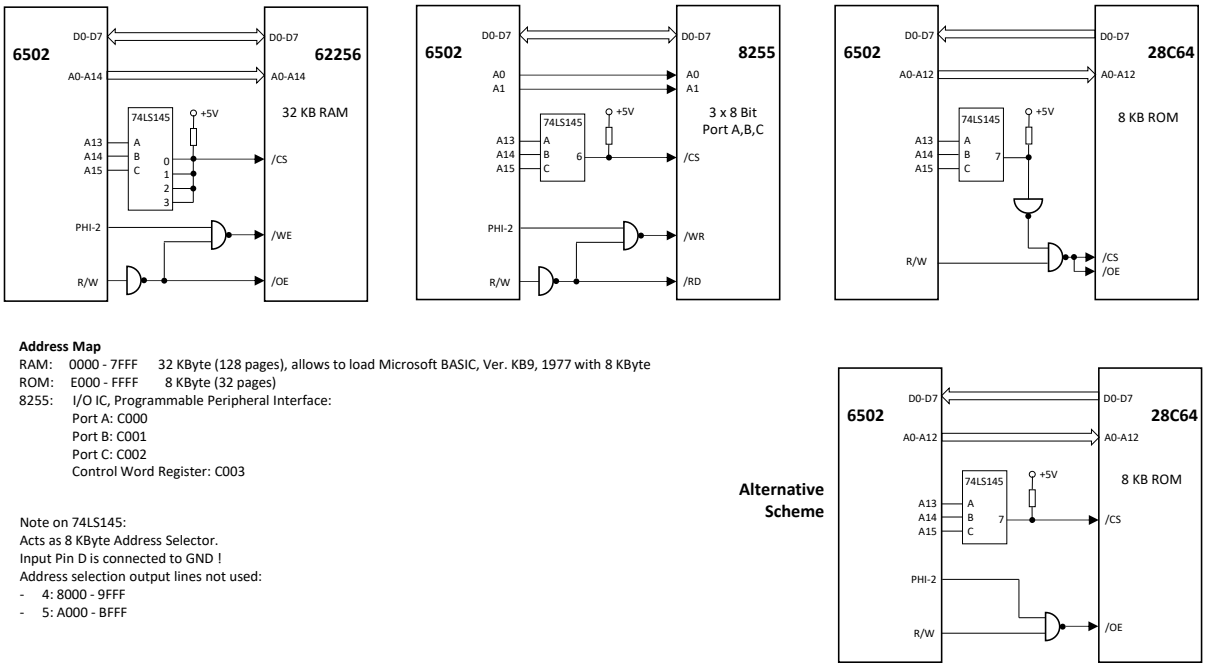


Figure 2-1: My-KIM Functional Block-Diagram



Address Map
RAM: 0000 - 7FFF 32 KByte (128 pages), allows to load Microsoft BASIC, Ver. KB9, 1977 with 8 KByte
ROM: E000 - FFFF 8 KByte (32 pages)
8255: I/O IC, Programmable Peripheral Interface:
Port A: C000
Port B: C001
Port C: C002
Control Word Register: C003

Note on 74LS145:
Acts as 8 KByte Address Selector.
Input Pin D is connected to GND !
Address selection output lines not used:
- 4: 8000 - 9FFF
- 5: A000 - BFFF

Alternative Scheme

Figure 2-2: My-KIM Memory and I/O Control Logic

The favoured solution would have been to design the entire system on a single board, like KIM-1, but I am using the education version of the EAGLE CAD software (light version). This version only allows to design boards with a maximum size of 100 x 80 mm. The consequence was, that the entire design had to be divided into five different boards with the disadvantage of having several connectors to interface the individual boards, as shown in Figure 2-3.

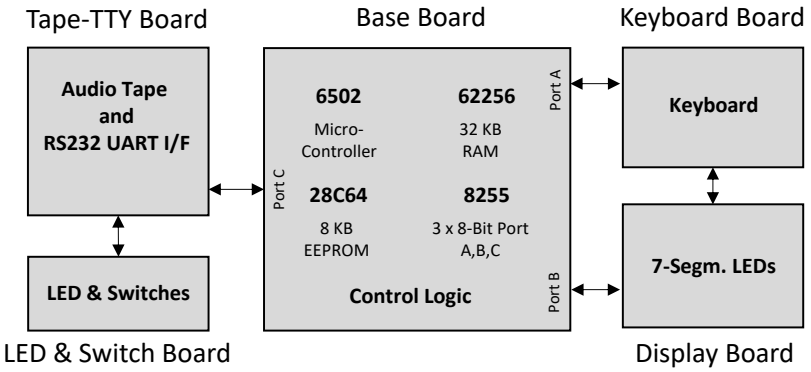


Figure 2-3: Five Dedicated My-KIM Boards

The schematics of the four hardware boards are shown in the following chapter (see Figure 2-4 to Figure 2-8).

2.2 Base Board

On the Base board the 6502 processor is driven by a 1 MHz crystal, buffered by a 74LS04 inverter (IC2A). In this design the C11 capacitor with 10 pF at the PHI2 output is absolutely necessary to achieve a clean clock pulse signal. The three interrupt input lines RESET, NMI and IRQ are pulled-up by 3.3 KΩ resistors. In addition, the ready line (RDY) is pulled-up, which allows to operate the processor in a single step mode (SST), i.e. one instruction at a time. The connector JP3 allows to create IRQ interrupts, but I have so far not tested this feature. The symbol used for the 6502

processor is based on a one from the W65C02 IC, because I didn't feel like designing an own symbol for the 6502 processor in EAGLE.

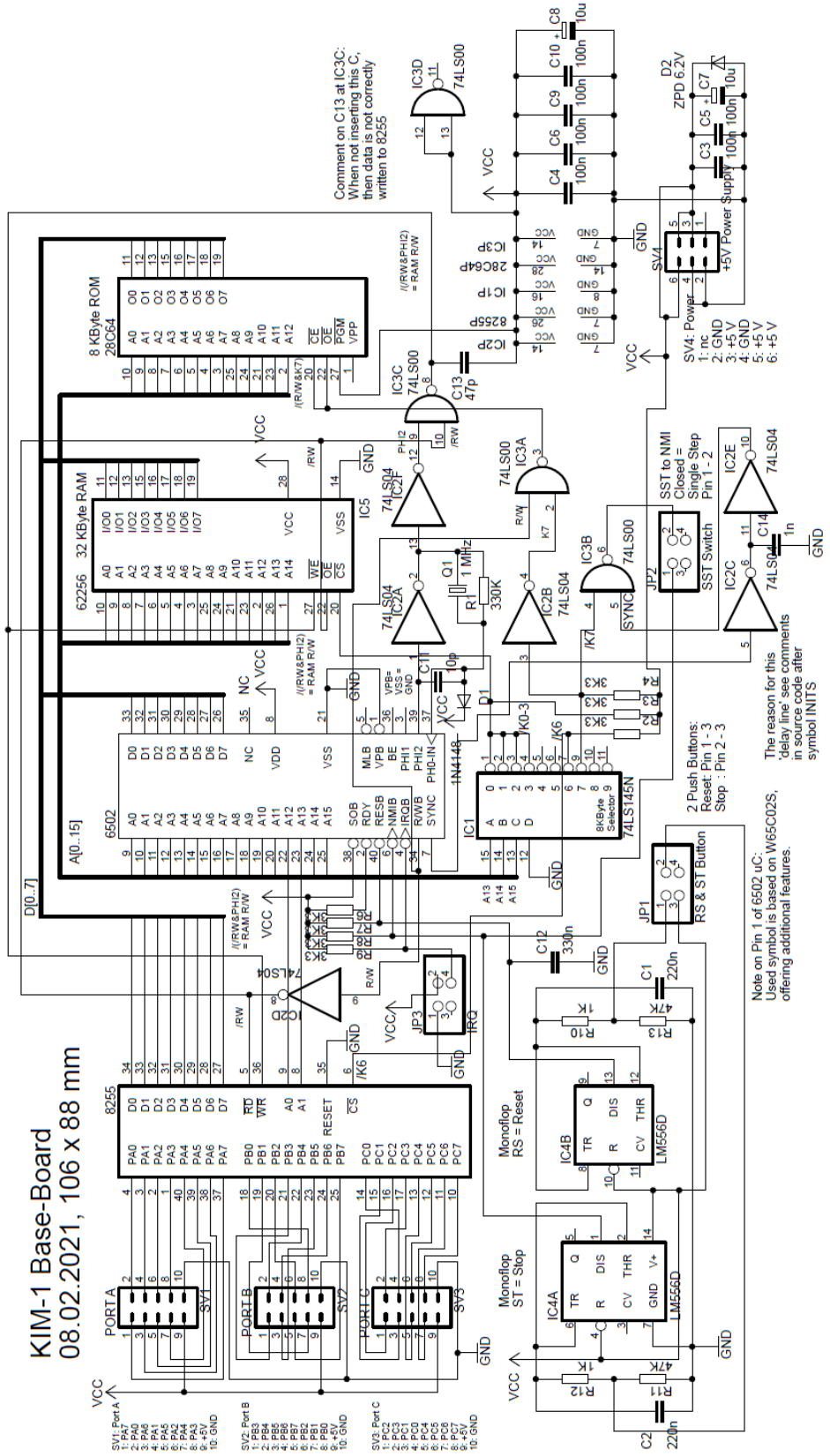


Figure 2-4: Base Board Schematics

For security I added a 6.2 V Zener diode D2 in the power supply chain, because you never know what happens during testing, when the first beer has been drunken. The used dual timers of NE556 are working as mono-flops, generating a negative pulse for resetting the processor, [RS] key, or to stop a running program, [ST] key. Having built up everything, it turned out that the 8255 PPI didn't work correctly. I therefore inserted a 47 pF capacitor at the output of the NAND gate IC3C, which solved the problem.

I added the capacitor C12 to the RST input of the processor, which should hold down this input line for some micro seconds to generate a reset during power on. However it turned out, that this does not work and you always have to push the [RS] button to generate this signal manually.

For the generation of the NMI interrupts for the single step mode the NAND gate IC3B is driven by two signals:

- the SYNC signal of the 6502 processor and
- the 74LS145 decoder output address line 7 (pin 9) for selecting the ROM area

The logic of this NAND gate is as follows:

- When a non-EEPROM address is present (pin 9 of IC1=74LS145 is high) and
- the SYNC signal becomes high for fetching an opcode

then the output of the NAND gate shall generate a low level signal at NMI interrupt input line, which in turn requests the processor to jump to the processor EEPROM address $FFFA, which is pointing to the RAM address $EC2F.

Because the 74LS145 has open collector outputs, it does not switch so fast like the standard TTL ICs with a totem pole transistor output network. In addition, the 74LS145 has a propagation delay time of up to 50 ns.

The result is, that the SYNC high level signal reaches the NAND gate ca. 40 ns earlier than the output of pin 9 of 74LS145. This creates for EVERY opcode fetch (independent of the current address) a small negative spike with a duration of 10-15 ns at the output of the NAND gate, hence the NMI interrupt is executed also when the processor is loading an opcode from the EEPROM, and this is not what we wanted, because the NMI interrupt shall only be executed for non-EEPROM addresses !

It was therefore necessary to delay the SYNC signal by adding two 74LS04 inverters as shown in the schematics (IC2C and IC2E). The inserted capacitor of 1 nF delays the SYNC signal by approximately > 25 ns, which is sufficient to avoid the small negative spike at the output of the NAND gate IC3B.

With these design modifications the 6502 processor runs absolutely stable in both modes, the free running one and the single step mode ! This however does not answer the question, why this single step problem didn't happen in the original KIM-1 hardware design, although I used the same parts and the same schematics.

2.3 Audio Tape and TTY Board

This board includes three functions:

- The audio tape write and read function
- The RS232 UART serial interface
- A DC-DC up-converter to generate the 12 V power supply for the audio tape ICs

The audio tape design is a 1-to-1 copy of the KIM-1 design with exactly the same ICs, namely the LM565 PLL circuit and the LM311 comparator. With the 5 KΩ potentiometer R18 we can set the free running PLL oscillator frequency to the required value of ca. 2765 Hz. Placing the potentiometer to the mid position is already a good starting point. The firmware in My-KIM offers a suitable small calibration

program to correctly set this potentiometer. The jumper JP2 allows to select either the high or the low level audio tape output signal (high = recorder line input, low = recorder microphone input).

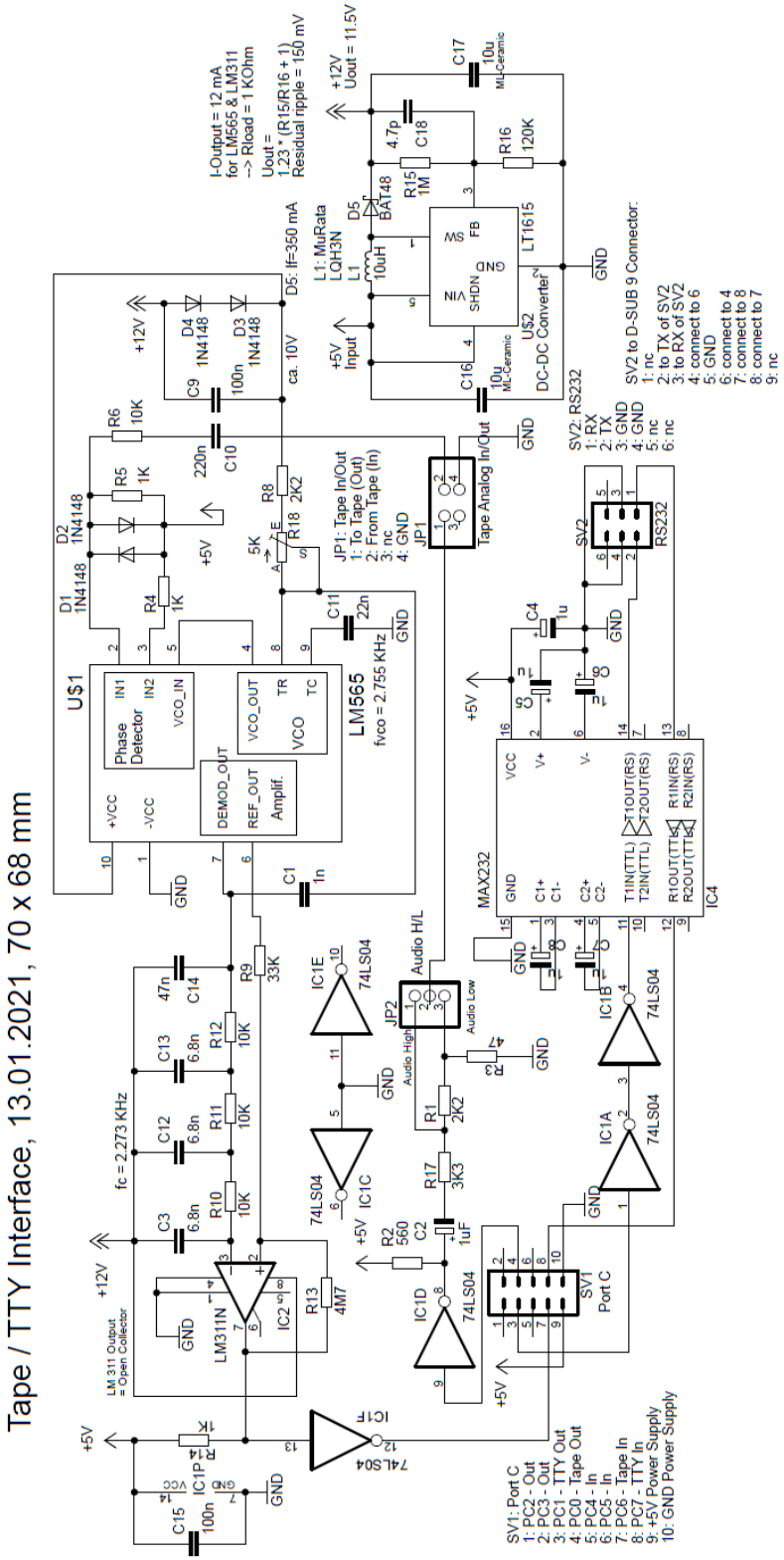
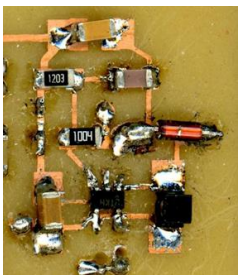


Figure 2-5: Audio Tape and TTY Board Schematics

To connect My-KIM to the PC, a RS232 UART serial interface on the basis of the MAX232 is implemented. However, this is only possible, because we are driving the 6502 processor by a crystal. A non-crystal driven system would only produce a garbage data stream at the PC side. The output of the MAX232 (SV2) is connected to a standard D-SUB 9 connector.

The two analogue ICs of the audio tape receive chain need a supply voltage of 12 V, but with only 12 mA. To avoid a dedicated power supply, I implemented a cheap DC-DC up-converter on the basis of the LT1615 IC. With the given dimensioning, the converter work very accurate and stable, having a final voltage ripple of only 150 mV peak-to-peak, which doesn't harm the PLL.



BBBUUUTTTT: The LT1615 comes in an SOT 23-5 outline, which requires a more than very steady hand and great glasses when soldering. The best is to do this work before you have your first cup of coffee in the morning. In addition, it is absolutely necessary to build up the entire electronics on the basis of SMDs (I use 1206 outlines, which are fairly good to solder manually) and to use high quality multi-layer ceramic capacitors (10 µF) and a 10 µH inductor of type LQH3N from Murata. The used Shottky diode is of type BAT48, which offers the required forward current of up to 350 mA. When you don't consider these guidelines, the up-converter doesn't work and only generates all kind of wild frequencies.

Based on this power supply, the audio tape receiving chain works very precisely. I tested it by storing the audio output signal on a smartphone (use an usual audio recording App) and then replayed the recorded audio signal, see Figure 5-6. The board has to be connected by SV1 to the Base board at port C of the 8255 PPI. This connector also provides the 5V power supply for this board.

2.4 Display Board

The Display board design, see Figure 2-6, is also a 1-to-1 copy of KIM-1. I used 14 mm common anode 7-segment LEDs to avoid always wearing glasses. I replaced the single transistors with separate driving resistors in the KIM-1 design by digital BCR158 SMD transistors, having already integrated these resistors.

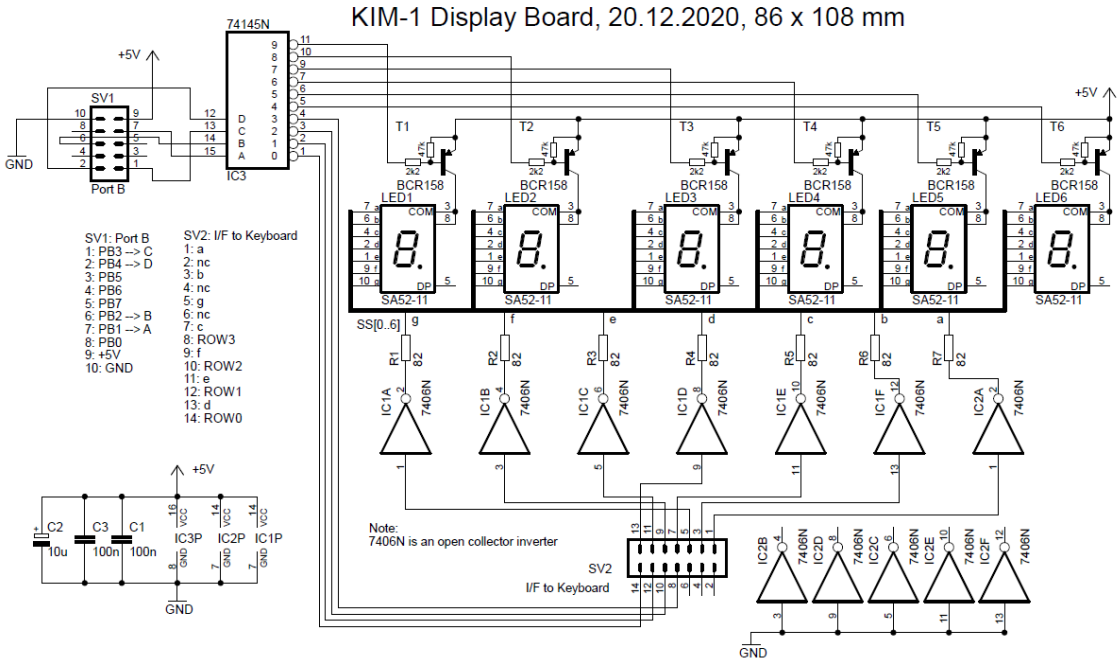


Figure 2-6: 7-Segment Display Board Schematics

The used 82 Ω current limiting resistors are a good choice, because the LEDs are driven by a multiplexed signal, as generated by the 6502 processor firmware. The board is directly connected via SV1 to the Base board at the 8255 port B by a ribbon cable. This connector also provides the necessary power supply of 5V. The only drop of bitterness is the additional 14-pin connector SV2 to the Keyboard board, which is due to the limiting size of the four individual boards.

2.5 Keyboard Board

The Keyboard board, see Figure 2-7, is straight forward and implements 23 push buttons. The push button matrix is electrically fully identical with the ones from the KIM-1 design. In addition, it features the TTY/KB switch, which allows to operate My-KIM either in the Keyboard or the TTY mode. Like the Display board, this board has to be interfaced with this board by the connector SV2. The connector SV1 has to be connected to the Base board at port A of the 8255 PPI.

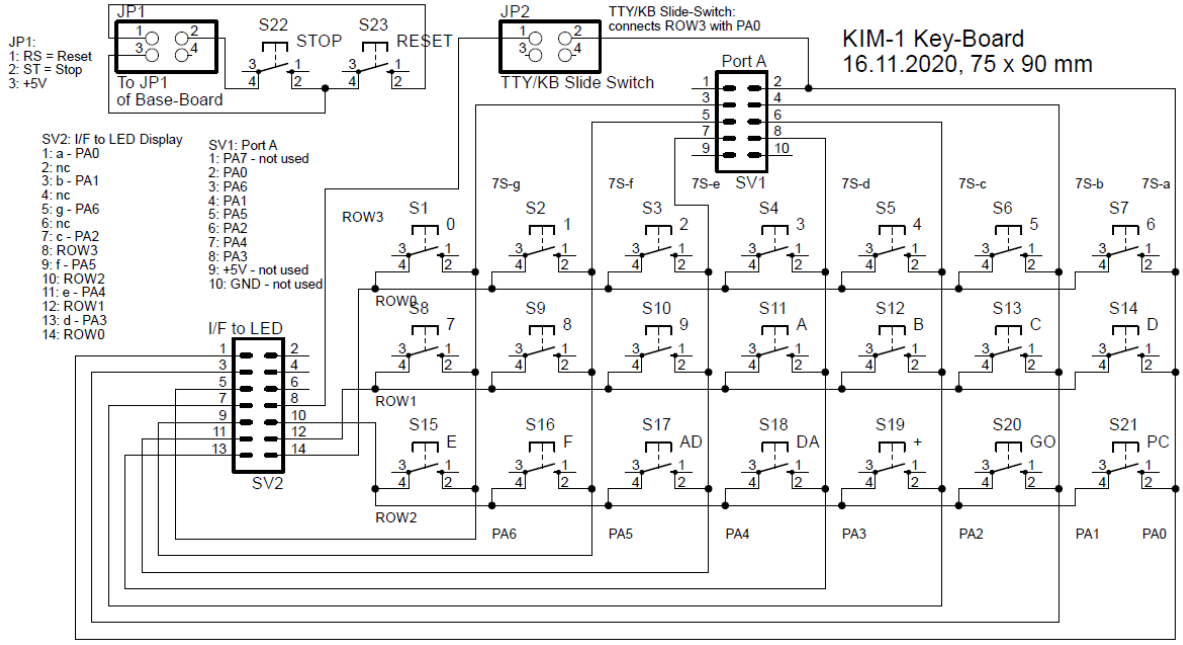


Figure 2-7: Keyboard Board Schematics

2.6 LED and Switch Interface Board

To allow a minimum communication level of My-KIM with the outer world, an additional LED and Switch interface board has been designed, which is connected to the free lines of port C of the 8255 PPI, see Figure 2-8. It uses the following port lines:

- Port C2 and C3 as output, driving two LEDs
- Port C4 and C5 as input, controlled by two switches

For driving the LEDs we need currents of up to 20 mA. Because the port C lines of the PPI only offer drive currents of up to 4 mA, additional 74LS04 (SMD type) inverters have been included, see JP3. The input port lines are pulled up to +5V by 4K7 resistors, which can be pulled down to GND by two switches, see JP2. The additional connector JP4 allows to drive externally connected electronics, if required. These output pin are protected against short cuts by 270 Ohm resistors. All resistors are based on 1206 SMDs, also the 74LS04 is an SMD device.

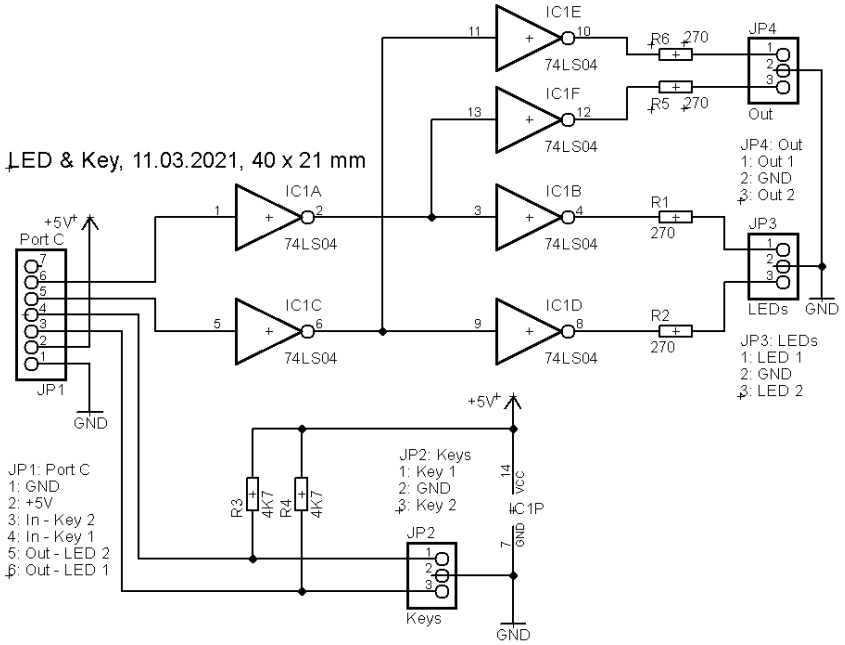


Figure 2-8: Additional LED and Switch (Key) Interface

2.7 EEPROM Programmer Board

The EEPROM programmer board is designed to program 8 KByte EEPROM ICs of type 28C64, see Figure 2-9. I consist of an ATmega8 controller, running with a crystal frequency of 10 MHz. Because the ATmega8 doesn't have enough output lines to fully control all EEPROM input lines (A0-A12 and D0-D7), it was necessary to realise the address control lines by a shift register based on 2 x 74HC595. The data lines D0-D7 are directly driven by the ATmega8 output ports. The serial PC interface for commanding the ATmega8 and programming the EEPROM is realised by a MAX232 IC, which is directly connected to a serial D-SUB 9 connector. The connector SV1 allows to flash the firmware for the ATmega8 directly from the Atmel Studio 7. The jack J1 provides the programmer with a regulated 5V power supply. Also here I added a 6.2V Zener diode in the power supply chain, you never know.

Because I don't use the RDY/BUSY output line (pin 1) of the EEPROM to identify the completion of a write cycle, the ATmega8 uses a delay period of ca. 6 ms before the next write cycle is initiated.

The programmer works very reliable and was tested with new Atmel (Microchip) AT28C64 devices as well with older ICs of type M28C64C from STMicroelectronics, manufactured in the late 90th. The programmer is driven by the 6502 IDE via the serial UART interface.

It is self-explaining, that the EEPROM programmer is placed in a separate housing and not into the dedicated My-KIM housing, as shown in the next chapter.

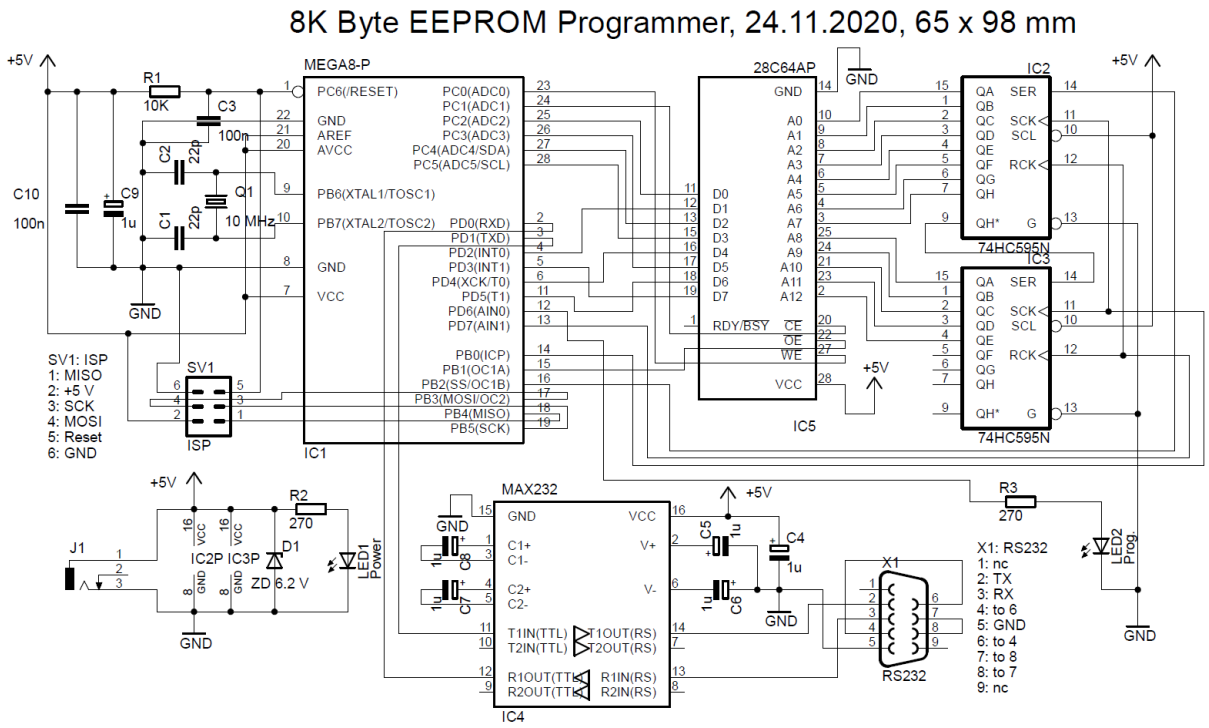
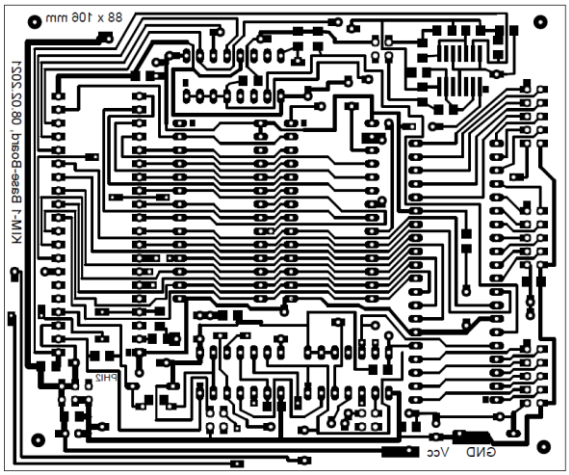


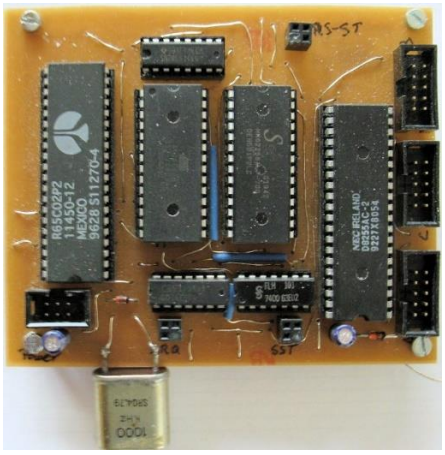
Figure 2-9: EEPROM Programmer Board Schematics

2.8 Manufacturing the Boards

The layout of the boards were designed with EAGLE. Because of the complexity of the circuits, it was not possible to design them as single layer boards and I was therefore required to create double sided boards. However, only the bottom layers were printed, while the minor upper layer paths were hand-wired by 0.4 mm silvered copper wires. The bottom layer paths as well as the fully equipped boards are shown in the figures below.

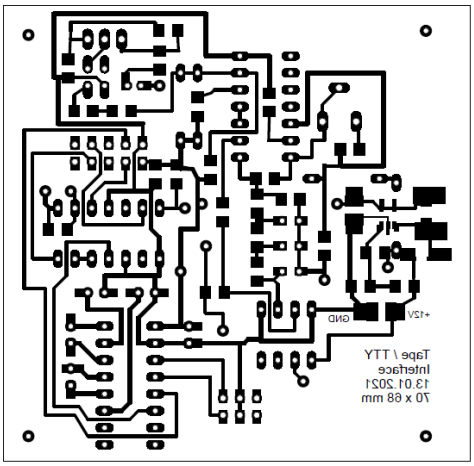
The 1 MHz crystal on the Base board was foreseen as an HC-18 outline. Having built up everything, it turned out, that this type of crystal is more than difficult to purchase. I therefore took an old bulky one, having nearly the double size. This looks very ugly, but it works. If someone knows a reliable source for 1 MHz crystals in an HC-18 housing, I would be happy to get the coordinates of the provider.

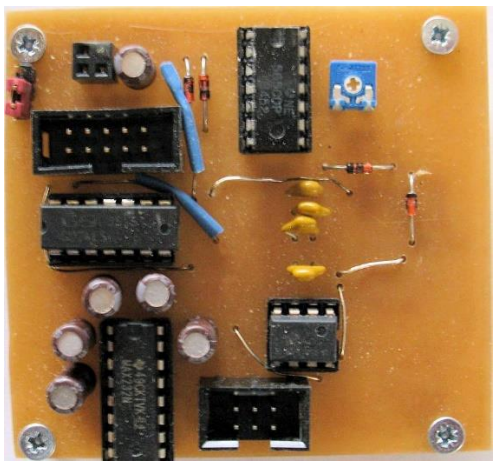




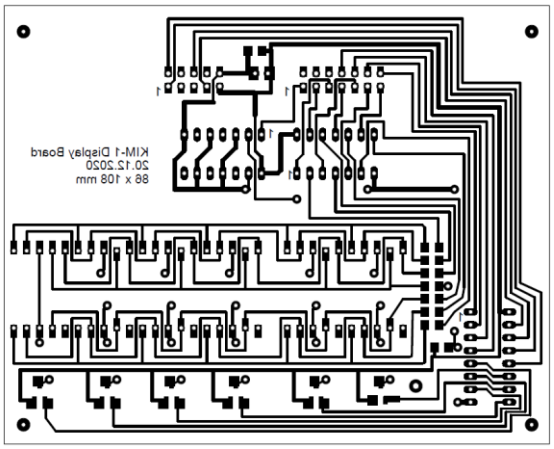
Hardware

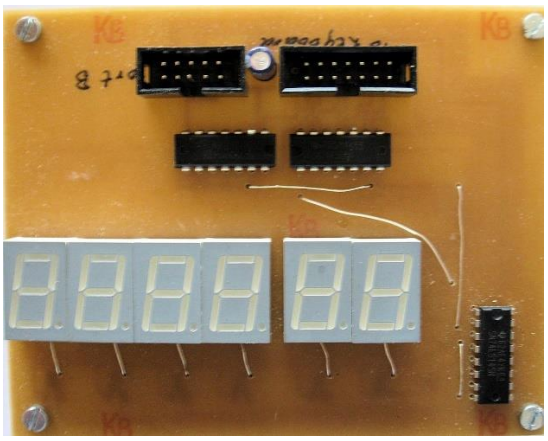
Base Board



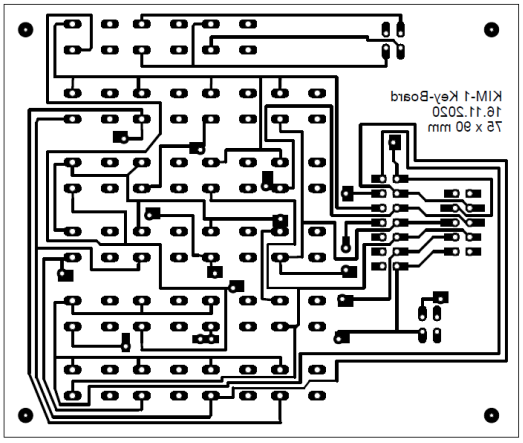


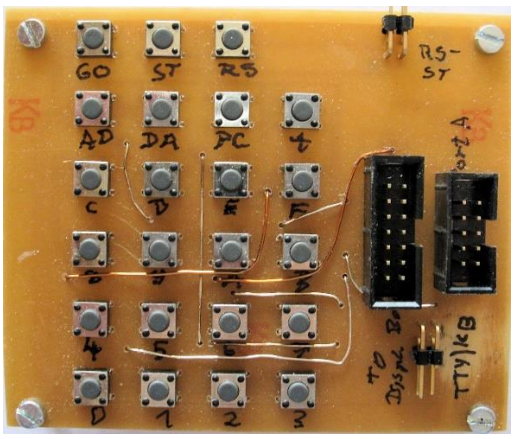
Audio Tape & TTY Board



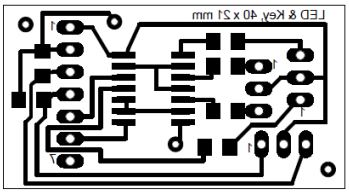


Display Board

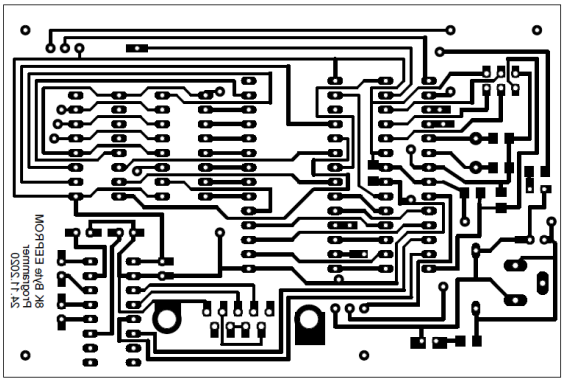


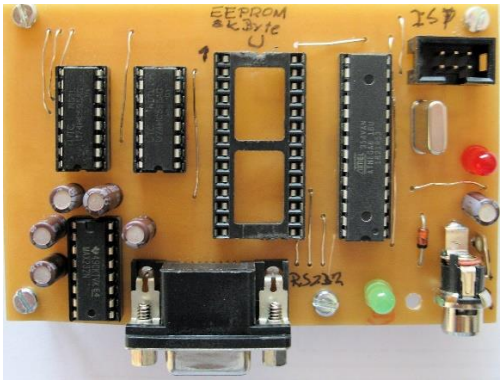


Keyboard Board



Additional LED and Switch Interface





EEPROM Programmer Board

Figure 2-10: Bottom Layer and fully equipped Boards

As can be seen, nearly all resistors and capacitors are based on 1206 SMDs, which allows to keep the boards as small as possible, especially for the Base board. The ICs are based on DIP outlines with the exception of the NE556 dual timer on the Base board, which comes with an SO-14 SMD outline. To allow to exchange the DIP ICs, suitable sockets were soldered. For the EEPROM I suggest to use a high quality socket, because it may be necessary to remove or insert the IC several times, in particular for the EEPROM programmer. This is what I haven't done, as you can see in the above picture. As already mentioned above, the components of the DC-DC up-converter on the Audio Tape & TTY board are realised by SMDs.

There are several poor men 4-pin connectors. To avoid a wrong plug in of these connectors, one pin of the female connector is filled with glue, while the corresponding pin of the male connectors is cut.

Figure 2-11 shows the configuration setup during firmware development with the flying ribbon cables, which are still too long, connecting the boards to one another. This allows to easily dismount the EEPROM on the Base board for any new firmware update.

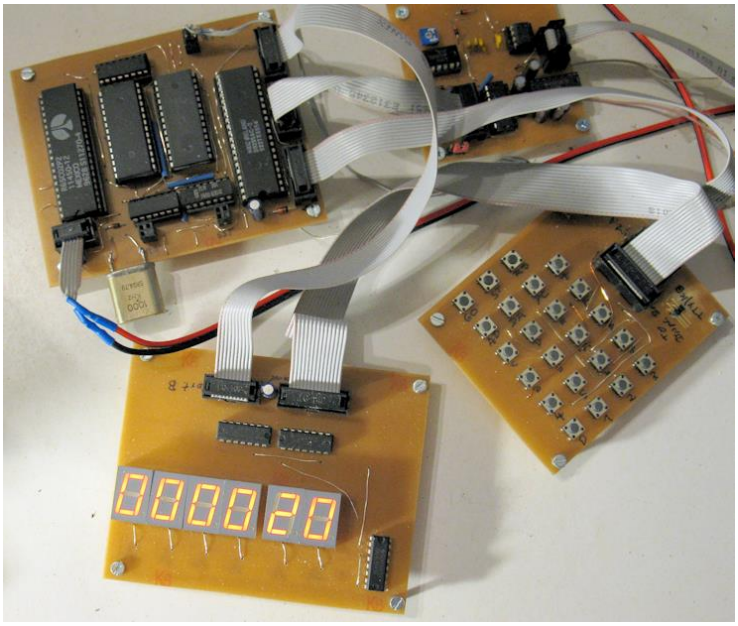


Figure 2-11: Test Configuration during My-KIM Operating Software Development

2.9 Housing and Mechanical Push Button Field

When working with My-KIM, it is definitively helpful to place the boards into a housing, allowing to have a cup of coffee in one hand, while typing with the other hand during software development. The housing is made out of 3 mm white FOREX rigid foam panels, which are very easy to cut, to glue and to paint. The side walls are glued to the bottom panel and the electronic boards are screwed on prepositioned rigid foam pads at the bottom panel. The power supply (regulated 5V DC), the RS-232 (D-SUB 9) and the Audio Tape Input / Output (cinch) connectors are mounted on the backside of the housing.

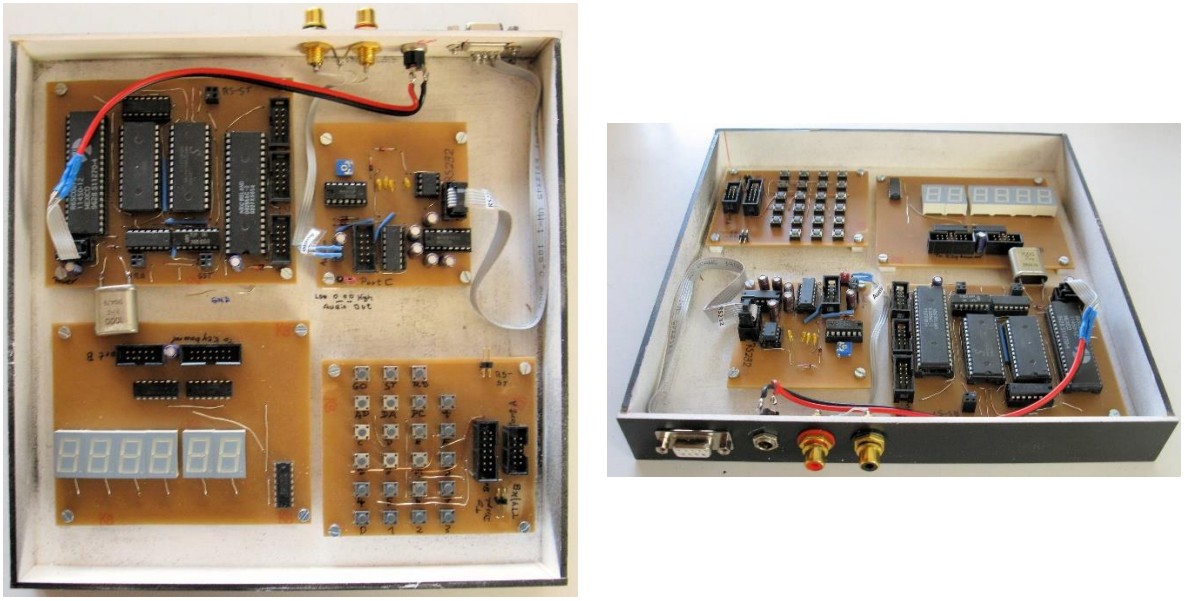


Figure 2-12: Opened Housing and Internal Layout without Inter-Board Cabling

A real challenge was the reconstruction of the KIM-1 keyboard. Commercial ones do either not have the required number of buttons, do not have the correct arrangements or are too small or too large. But in most cases they are far beyond my financial potentials. The final choice was to manufacture a

dedicated push-button field with my 3D printer. The result is shown in the figures below, consisting of an upper cover plate (blue), 23 quadratic push button caps (yellow) and two distance plates (green) to arrange a kind of guidance for the cylindrical extension rods (red) between the keys on the Keyboard and the push button caps. The two plates are glued together with two 5 mm rigid foam distance brackets. This arrangement allows to keep the rods as 'free hanging' parts, without any fixation on either the keyboard or the push button cap side. The only pre-requisite is, that they all shall have exactly the same length. With the exception of the cylindrical rods, which are made out of 4 mm aluminium, all parts are 3D printed with black PLA.

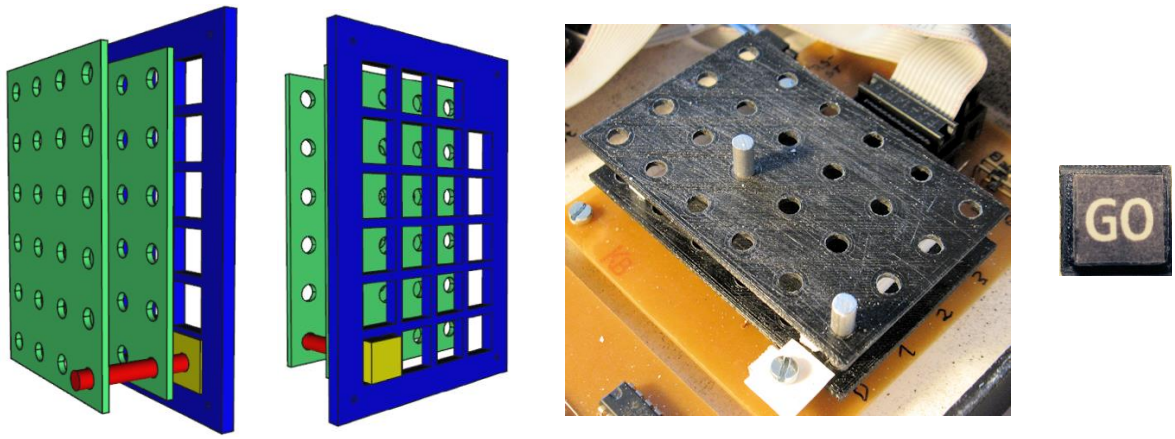


Figure 2-13: DIY 3D Printed My-KIM Push-Button Field

The top panel is also made out of 3 mm FOREX rigid foam. Having made the 7-segment display cut-out and the drill holes for the LEDs and the switches, the plate has been sanded, sprayed with grey colour paint and finally sprayed with clear coat to protect it from scratches. The panel is fixated to the black painted side walls of the housing by additional rigid foam brackets, which allows to use small 2 mm recessed head screws to mount the top panel.

Figure 2-14: Completed My-KIM Housing including Annotation Labels

The display frame (also 3D printed) and the key cover plate were sprayed with red paint to give the case a little bit more colour. Whether you like this colour is a matter of taste, as is usual in life. The

annotation labels on the top panel and on the push button caps were printed on self-adhesive paper, which are in addition protected from the top with self-adhesive book cover transparency film. After this procedure they are cut-out with a sharp scissors and then glued on the foreseen places.

However I have to note something at the end: designing and building the housing and the push-button field, in particular the sophisticated three dimensional cut-out for the display and putting everything together took more time, than designing and testing the electronic boards !

3 My-KIM 6502 Firmware

3.1 Original KIM-1 Firmware

In a first step I searched in the internet for the original KIM-1 ROM firmware, but most of them were incomplete or erroneous. The final choice was therefore to go line by line through the source code in the appendix 1 of the “KIM-1 User Manual”. The completed and correct assembler source code comes with several comments to improve the readability of the software. This assembler file is called “KIM-1-Source-Code.asm”.

3.2 6502 Integrated Development Environment

Having completed the KIM-1 source code doesn’t really help, because we need a suitable assembler to create the binary code to be flashed to the EEPROM. In the early 90th I found a 6502 assembler source code, already written in C. The author was:

> George V. Wilder, Berkeley University
> Address: IX 1A-360 x1937, ihuxp!gvw1

Today I tried to find him in the internet, to get his permission to use it for private purposes, but I didn’t succeed, I only found some correspondences covering CPM programming issues out of the years 1984/1987 like:

> www.retroarchive.org/cpm/cdrom/SIMTEL/ARCHIVES/CPM/8408-1.TXT
> www.columbia.edu › apxa2

Therefore I clearly state here, that the 6502 core assembler was written by George V. Wilder, Berkeley University. I extended this UNIX like console assembler to a window based 6502 Integrated Development Environment (IDE), offering the following features:

- Loading, editing and saving 6502 assembler source code files
- Assembling the source code and generating Intel Hex files as well as List files
- Debugging the source code including break point definitions
- Showing and editing the total 6502 memory area
- Flashing the binary code to an 8 KByte EEPROM

The result of this software development is described in chapter 5. It shall be mentioned, that the assembler can only be used for the early 6502 processors, but not for the 65C02 successor, having additional instructions not supported by this assembler.

3.3 Necessary Changes in the Original KIM-1 Source Code

By means of the developed IDE I was able to update the KIM-1 source code to the My-KIM hardware environment. Major firmware updates due to hardware changes were in the following areas:

- The KIM-1 firmware is located in two ROM devices with an address range of 0x1800-0x1BFF (6530-003) and 0x1C00-0x1FFF (6530-002). These addresses had to be changed to the My-KIM address range 0xE000-0xFFFF.
- The KIM-1 operating system uses the free RAM area in the 6530-002 IC (0x17E7-0x17FF). My-KIM offers 32 KByte RAM (0x0000-0x7FFF). Hence these KIM-1 RAM addresses had to be converted to 0x7FE7-07FFF.
- KIM-1 uses the 8-bit ports A and B of the 6530-002 IC. These two ports drive the 7-Segment LED display, read the key of the keyboard, drives/reads the audio tape and the TTY input/output lines. In My-KIM we have the 8255 PPI IC with the addresses 0xC000=Port A, 0xC001=Port B, 0xC002=Port A and 0xC003 for the 8255 Control Word. These ports are used as follows:
  - Port A works as input to read the keys or works as output to drive the 7-Segment LEDs
  - Port B drives the 74LS145 selector IC for the 7-Segment LED display and the keys

- Port C is used as audio tape and TTY input/output (C0=Tape Out, C6=Tape In, C1=TTY Out, C7=TTY In).

Hence all port addresses and their bit positions had to be changed.

- KIM-1 uses the timer in the 6530-003 device to generate the audio tape data stream. Because My-KIM doesn't have a hardware timer, the timer functionality is realised in My-KIM by a dedicated, but accurate delay software (subroutine WAIT).

Beside the hardware based changes I added additional software for the communication with the PC via the UART (MAX232). These software add-ons allow to transmit complete RAM areas to the PC or vice versa, using a communication protocol adopted to the Intel Hex file format, including checksum error detection. Based on these software add-ons it is e.g. possible to save and load user written codes to a file on PC and to load the Microsoft BASIC interpreter in binary file format.

All changes made to the original KIM-1 firmware are documented in the related My-KIM assembler source code, which needs only 3 KByte in the EEPROM, hence there is enough space for further firmware extensions. This assembler source code file is named My-KIM_V1.asm, the associated Intel Hex file is called My-KIM_V1.hex. The resulting My-KIM memory map is as follows:

Table 3-1: My-KIM Memory Map

Type	Address	Description
RAM	$0000 - $7FFF	32 KByte (128 pages), allows to load Microsoft BASIC, V1.1, KB9, 1977
ROM	$E000 - $FFFF	8 KByte (32 pages)
I/O Device Intel 8255	$C000 - $C003	Port A address = $C000, used for Keyboard, 7-Seg. LEDs and TTY/KB bridge Port B address = $C001, used for 74LS145 Row and 7-Seg. LED Selector Port C address = $C002, used for Tape In/Out, TTY (UART) In/Out 8255 Control Word address = $C003

The following RAM address ranges are occupied by the My-KIM operating system:

$00EC - $00FF	Page Zero:	holding Variables
$0100 - $01FF	Page 1:	holding the 6502 Stack
$7FE5 - $7FFF	Page 127:	holding Variables

These memory areas shall not be used by any user program and their content shall not be modified. If you do so, then the operating system may crash.

3.4 Necessary Changes in the Microsoft BASIC Binary Code

The Microsoft BASIC interpreter for KIM-1 comes as a binary file, named KB9.BIN. By chance I found the original source code in the internet, which was written on a PDP-10, using the MACRO-10 assembler, see source code snippet example in Figure 3-1. Of particular interest are the macro op-odes (e.g. LDWX), generating two 6502 op-code instructions (e.g. LDWX → LDX TXTTAB and LDA TXTTAB+1). Only based on this documentation I was able to customise the necessary patches for the BASIC binary code and to adapt it to the My-KIM hardware and software environment.

```
FNDLIN:   LDWX   TXTTAB    ; LOAD [X,A] WITH [TXTTAB]
FNDLNC:   LDYI   1
          STWX   LOWTR     ; STORE [X,A] INTO LOWTR
          LDADY  LOWTR     ; SEE IF LINK IS 0
          BEQ    FLINRT
          INY
          INY
          LDA    LINNUM+1  ; COMP HIGH ORDERS OF LINE NUMBERS.
          CMPDY  LOWTR
          BCC    FLNRTS    ; NO SUCH LINE NUMBER.
          BEQ    FNDLO1
          DEY
```

Figure 3-1: Source Code Snippet of the Microsoft BASIC Interpreter for KIM-1

Major patches were necessary for the TTY / UART input and output subroutine addresses (GETCH, OUTCH symbols), for the LOAD and SAVE commands regarding the start and end addresses from where to read and where to write a user written BASIC program. BASIC also uses some addresses in the RAM Volatile Execution Block (VEB) of the KIM-1 operating system, which had to be updated. Finally we have to make a patch regarding the recognition of the so-called CONTROL-C key to halt a running BASIC program or to stop the listing of a BASIC program. All patches will be performed by the My-KIM operating system after having loaded the BASIC interpreter binary file into the RAM. This means, that the original KB9.BIN file has not been altered.

Based on these patches it is possible to load and start the Microsoft BASIC interpreter in the self-developed IDE, which among others allows to load and save user written BASIC program to or from the PC in files via the UART interface with user defined filenames.

In addition, BASIC allows to perform user written assembler code residing in the free RAM area ($0200 - $1FFF) by calling the function USR(). For details on this, see chapter 7.8.

4 Using My-KIM as Stand-Alone System

4.1 Overview

My-KIM can be used as stand-alone system without any connection to a PC. In this mode, My-KIM is operated by the integrated keyboard, supported by the 7-segment displays. The keyboard features 16 hexa-decimal push buttons [0] … [F] and seven control buttons [GO] … [+]. The use of the control buttons is described in the next chapter.

The display is divided into an address field and a data field. The address field has four digits, pointing to a memory address in the range from $0000 to $FFFF. The two digit data field shows the current content of the addressed memory field represented as hexa-decimal byte, ranging from $00 to $FF.

In this stand-alone mode, the “Mode” switch shall always be in the position “KB”, which stands for Keyboard. The SST switch (Single Step) shall initially be in the position OFF.

There are two additional switches and LEDs, which can be used as input and output indicators, to allow a program to check the state of a switch and to show the status of a program indicator. The use of these are described in chapter 4.3.

Figure 4-1: The Keyboard and the Display with 2 Control LEDs and Switches

4.2 Operating My-KIM

Before applying power to My-KIM, the two switches shall be put to the following positions:

- Single Step (SST) = OFF
- Mode = KB (Keyboard)

After power ON, you first have to push the [RS] button (RESET), which will light up the display. It shows the current memory address, which is $0000 and its content in hexa-decimal values, which is an arbitrary value at the beginning. To change the address, push the [AD] button (ADDRESS) followed by pushing the hexa-decimal address keys [0] … [F] to the address, you want to set. If you want to increment the displayed address by 1, simply push the [+] button. To change the content of a memory address, push the [DA] button (DATA) followed by entering the byte value, this address shall become.

Also in this DATA mode you can push the [+] button to change to the next higher address to be displayed. With these buttons you are ready to enter the program, which counts up to 5, as shown below. When the program has counted up to 5, it will jump back to the My-KIM operating system, which will light up the display. In the stand-alone system you should always use for this purpose the address $EC6B.

Table 4-1: Program 1 – Counting up to 5

Address	OP-Codes	Assembler Language Equivalent		
0200	18		CLC	; Clear Carry
0201	A9 00		LDA #0	; Set accu to 0
0203	69 01	LOOP	ADC #1	; Add 1 to content of accu
0205	C9 05		CMP #5	; Compare accu with 5 (accu – 5)
0207	F0 FA		BNE LOOP	; Branch to LOOP if Z flag is not 0 (accu – 5 is not 0)
0209	4C 6B EC		JMP START	; Jump back to My-KIM operating system (Display ON)

Having typed-in this small program, set the address back to “0200 18” and then push the [GO] button. The display will go off for a fraction of a second and will then light up with a display of “0209 4C”. This means, that the processor has counted up to 5 and then jumped back to the operating system by the last operation performed, which was “4C 6B EC”.

This of course doesn’t really show, what happened. We therefore put the SST switch to the position ON, allowing to perform the program in single instruction steps. To do so, set the address back to “0200 18” and then type the [GO] button. The processor will now perform the “CLC” instruction and will stop at the next instruction, which is “0201 A9”. By repeatedly typing the [GO] button, you can ‘walk’ through the program and you can check the correctness of your typed-in program.

Now we write a second program, which is shown below. This program also counts up to 5, but in addition, it saves the content of the accumulator to the memory, which is incremented by each loop by 2, to the memory locations $0300 and upwards.

Set the SST switch back to OFF, go to address “0200 18” and then press the [GO] button. When the display lights up, the program has completed its job and you now can check, that the memory locations $0300 to $0304 shall have the following contents: $0300=0, $0301=2, $0302=4, $0303=6, $0304=8.

Table 4-2: Program 2 – Counting up to 5 and Store Data

Address	OP-Codes	Assembler Language Equivalent		
0200	18		CLC	; Clear Carry
0201	A2 00		LDX #0	; Set X-register to 0
0203	A9 00		LDA #0	; Set accu to 0
0205	9D 00 03	LOOP	STA 0300,X	; Store accu content to address 0300+X
0208	69 02		ADC #2	; Add 2 to content of accu
020A	E8		INX	; Increment X-register (X=X+1)
020B	E0 05		CPX #5	; Compare X-register with 5 (X – 5)
020D	D0 F6		BNE LOOP	; Branch to LOOP if Z flag is not 0 (X – 5 is not 0)
020F	4C 6B EC		JMP START	; Jump back to My-KIM operating system (Display ON)

The next program 3, only modified at addresses $0308 and $030B, counts up to 255 = $FF hex. When typing [GO] (SST = OFF), the display will shortly go off while the processor is executing the program. Having finished, the display will light up and you can check, that the memory locations $0300 to $03FE shall have the following contents: $0300=0, $0301=1, $0302=2, $0303=3, … , $03FE=FE.

Table 4-3: Program 3 – Counting up to 255 and Store Data

Address	OP-Codes	Assembler Language Equivalent		
0200	18		CLC	; Clear Carry
0201	A2 00		LDX #0	; Set X-register to 0
0203	A9 00		LDA #0	; Set accu to 0
0205	9D 00 03	LOOP	STA 0300,X	; Store accu content to address 0300+X
0208	**69 01**		**ADC #1**	**; Add 1 to content of accu**
020A	E8		INX	; Increment X-register (X=X+1)
020B	**E0 FF**		**CPX #255**	**; Compare X-register with 255 (X – 255)**
020D	D0 F6		BNE LOOP	; Branch to LOOP if Z flag is not 0 (X – 255 is not 0)
020F	4C 6B EC		JMP START	; Jump back to My-KIM operating system (Display ON)

While the program is running (display is off), you can press the [ST] button (STOP), which terminates the program and the display will show the memory location, which should have been executed next.

The [PC] button (Program Counter) allows you to recall the value of the Program Counter at the time, when you have stopped the program. You may have performed a variety of operations since that time such as inspecting the contents of various machine registers stored at specific memory locations. When you press the [PC] key, the content of the Program Counter is recalled to the address field of the display. You now may continue program execution from that point by pressing the [GO] key.

4.3 Using the Two Control Switches and LEDs

The technical background of these two switches and LEDs is described in chapter 6.3. Here we show, how these control elements can be used within a program.

The program 4 shows how to perform a blinking of the LED 1. The program uses the WAIT subroutine, which is described in chapter 6.4. Set the SST switch to OFF and start the program at address $0200. Now the LED 1 shall blink with a frequency of about less than half a second. It shall be noted, that the program doesn't work in the single step mode, because the operating system always reset the Port C (and also the LED) due to the multiplexing the 7-segment display.

Table 4-4: Program 4 – LED 1 Blinking

Address	OP-Codes	Assembler Language Equivalent		
0200	20 9E F1	LOOP	JSR $F19E	; Set LED 1 to OFF
0203	20 A8 F1		JSR DELAY	; Delay for 0.25 seconds
0206	20 A3 F1		JSR $F1A3	; Set LED 1 to ON
0209	20 AD F1		JSR DELAY	; Delay for 0.25 seconds
020C	4C 00 02		JMP LOOP	; Jump back to address $0200
0240	A2 FF	DELAY	LDX #$FF	; Set X to 255
0242	8E F9 7F		STX $7FF9	; Store X to WAIT_TIME
0245	20 C1 F0	DELAY1	JSR WAIT	; Perform WAIT subroutine
0248	CA		DEX	; Decrement X by 1 (X=X-1)
0249	D0 FA		BNE DELAY1	; If X is not 0, branch to DELAY1
024A	60		RTS	; Return from subroutine

The next program 5 will read one of the two switches, which is switch 1. For reading the switch 1 we use the subroutine RPOC4 (Read Port C4) in the EEPROM, which is defined as follows:

Address	Op-Codes		Assembler Language Equivalent	
F1B5	AD 02 C0	RPOC4	LDA SCD	; Read Port C
F1B8	29 10		AND #$10	; Mask Bit 4
F1BA	60		RTS	

The subroutine RPOC4 returns with a zero flag (Z-Flag) = 0, when the switch is in position OFF. It returns with Z=1, when the switch is in position ON. Based on this we can write the program 5 for checking the status of this switch.

First put the SST switch to position OFF. Then place the switch 1 to position ON and start the program at address $0200 by pushing the [GO] button. As long as the switch 1 is in position ON, the program is continuously running. When you put the switch 1 to position OFF, then the program will terminate and jumps back to the operating system. This program can be used in the My-KIM stand-alone mode as well in the Teletype mode.

Table 4-5: Program 5 – Read Status of Switch 1

Address	OP-Codes	Assembler Language Equivalent		
0200	20 B5 F1	LOOP	JSR $F1B5	; Read switch 1 status
0203	D0 FB		BNE LOOP	; Branch to LOOP if Z flag is not equal 0
0210	4C 6B EC		JMP START	; Jump back to My-KIM operating system

When we want to check the status of the switch 2, we can use the subroutine RPOC5, see below.

Address	Op-Codes		Assembler Language Equivalent	
F1BB	AD 02 C0	RPOC5	LDA SCD	; Read Port C
F1BE	29 20		AND #$20	; Mask Bit 5
F1C0	60		RTS	

The below program 6 is similar, but works as follows: As long as the switch 2 is in position OFF, the program is continuously running. Set the switch 2 to position OFF and start the program at address $0200. When you put the switch 2 to position ON, then the program will terminate and jumps back to the operating system. Also this program can be used in the My-KIM stand-alone mode as well in the Teletype mode. You should recognise, that we are using here the “BEQ LOOP” statement in contrast to the “BNE LOOP” statement in program 5.

Table 4-6: Program 6 – Read Status of Switch 2

Address	OP-Codes	Assembler Language Equivalent		
0200	20 BB F1	LOOP	JSR $F1BB	; Read switch 2 status
0203	F0 FB		BEQ LOOP	; Branch to LOOP if Z flag is equal 0
0210	4C 6B EC		JMP START	; Jump back to My-KIM operating system

Based on program 5 and 6 we can modify the program 4 to work as follows:

- When switch 1 is in position ON, the LED 1 shall blink
- When switch 1 is in position OFF, the LED 1 shall not blink
- When switch 2 is set to position OFF, then the program shall terminate

First set both switches to position ON and start the program at address $0200. Now you can set switch 1 to position OFF and the blinking stops (LED lights up), but the program is still running. When you set the switch 2 to position OFF, then the program terminates and returns to the operating system.

You should also observe the following behaviour: When switch 1 is in position OFF (no blinking) and you put the switch 2 to position OFF, nothing happens, the LED is still ON, because the program still runs in the LOOP1 loop. Only when you put switch 1 to position ON, then the program terminates.

Table 4-7: Program 7 – LED 1 Blinking controlled by Switches

Address	OP-Codes	Assembler Language Equivalent		
0200	20 9E F1	LOOP	JSR $F19E	; Set LED 1 to OFF
0203	20 40 02		JSR DELAY	; Delay for 0.25 seconds
0206	20 A3 F1		JSR $F1A3	; Set LED 1 to ON
0209	20 40 02		JSR DELAY	; Delay for 0.25 seconds
020C	20 B5 F1	LOOP1	JSR $F1B5	; Read switch 1
020F	F0 FB		BEQ LOOP1	; Branch back to LOOP1, if switch 1 is OFF
0211	20 BB F1		JSR $F1BB	; Read switch 2
0214	D0 EA		BNE LOOP	; Branch back to $0200, if switch 2 is ON
0216	4C 6B EC		JMP START	; Jump back to My-KIM operating system
0240	A2 FF	DELAY	LDX #$FF	; Set X to 255
0242	8E F9 7F		STX $7FF9	; Store X to WAIT_TIME
0245	20 C1 F0	DELAY1	JSR WAIT	; Perform WAIT subroutine
0248	CA		DEX	; Decrement X by 1 (X=X-1)
0249	D0 FA		BNE DELAY1	; If X is not 0, branch to DELAY1
024A	60		RTS	; Return from subroutine

4.4 Dumping and Loading a Program to the Smartphone

Dumping and loading a program to/from the smartphone only works in the My-KIM stand-alone mode. Before using the following dump and load routines, make sure that the SST switch is in the position OFF. If you don't consider this, both programs will not work. Also make sure, that the jumper JP2 on the TTY / Audio Tape board is set to the "Low" position, which can only be done, when opening the housing of the My-KIM computer.

Dumping a Program

Having typed-in a program you may save it for later use. This can be achieved by dumping a program to the smartphone. The procedure is as follows:

- Store the program start address to the memory locations $7FF4 (SAL=low byte) and $7FF5 (SAH=high byte)
- Store the program end start address + 1 to the memory locations $7FF6 (low byte) and $7FF7 (high byte)
- Store the value $01 to the memory address $7FF8 (Program ID)
- Set the address field of the LED display to the address $E000
- Connect the backside cinch connectors by a suitable adapter cable to your smartphone
- Start a suitable audio App on the smartphone to record the data transmitted by My-KIM

Then push the [GO] button on the keyboard, which starts the transmission and switches the display off. Having completed the transmission, the display will switch on showing the address $0000 and you can stop the recording on the smartphone. Optionally you may give the audio recording on the smartphone a specific name.

Because the logical data rate is 8.3 Byte/s, it will take slightly more than 2 minutes to save 1024 Byte of a program to the smartphone. Same is true for loading a program.

Loading a Program

Loading a program from the smartphone is performed as follows:

- Store the value $00 to the memory address $7FF8 (Program ID)
- Set the address field of the LED display to the address $E073

Then push the [GO] button on the keyboard, which starts the loading of the program and switches the display off. Then start the replay of the audio file to be loaded from the smartphone. Having successfully loaded the program, the display will show the address $0000. In case of an error during the loading procedure, the display shows the address $FFFF.

The meaning of the used ID value for loading a program is as follows:

ID	Meaning
$00	Ignore ID and use Start Address on Audio Recording
$FF	Ignore ID and use SAL/SAH content as Start Address
$01 - $FE	Load program with exactly this ID and use Start Address on Audio Recording

The usual use of the ID in loading a program is ID = $00. If you want to load the program to a different address, set ID = $FF and set the SAL/SAH memory addresses to the required load address.

Warning: When loading a program, never load a program into the zero page area $00EC - $00FF, because there are several My-KIM operating system variables. Same is true for page 1 ($0100 - $01FF), which holds the stack of the operating system and the high RAM memory area $7FE5 - $7FFF. If you do so, the operating system crashes and does not respond anymore. This is true in the My-KIM stand-alone mode, when loading a program from the smartphone, as well as in the Teletype mode, loading a program from PC. Therefore: when writing a program, it is recommended, that you use the memory area from $0200 upwards.

The Jumper JP2

When you connect the cinch input / output lines to a PC audio card, set the JP2 jumper on the TTY / Audio Tape board to position “High” and select the “Line” input of the audio card of your PC. To do so, you have to open the housing of the My-KIM computer.

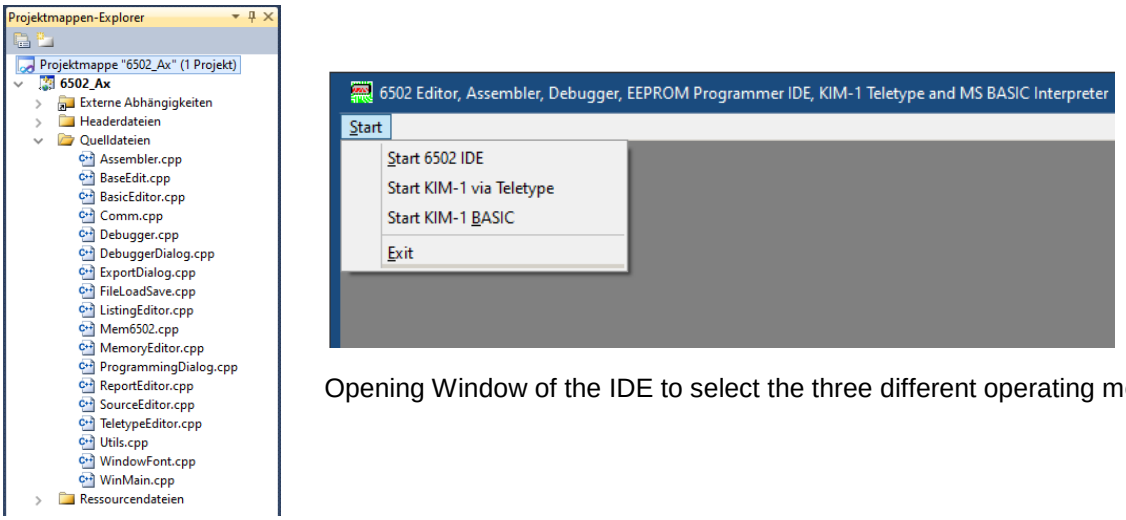
5 Integrated Development Environment

5.1 Overview

As already outlined in chapter 1, the Integrated Development Environment (IDE) allows to write and edit 6502 micro controller source code, assembling the code, debug the code and to flash the code to an 8 KByte EEPROM. In addition, the IDE allows to operate the My-KIM computer via its TTY / UART interface. Finally the IDE allows to load the Microsoft BASIC interpreter to the My-KIM RAM and to write and perform BASIC programs, which can be saved on the PC.

5.2 Architecture and Usage

The IDE architecture is based on a Windows Multiple Document Interface (MDI) application and has been developed with Microsoft Visual Studio. The software is written in C++ and comes with several .cpp and associated .h files as shown below in the left picture. Having started the IDE the user can select one of the three operating modes, as shown in the right below figure.



Opening Window of the IDE to select the three different operating modes

Figure 5-1: IDE Source Files and Opening Screen

5.3 IDE Mode

Starting the IDE mode will create a frame window with four MDI client windows as shown in Figure 5-2. To load a 6502 assembler file, to assemble it and to start a debugging session can be performed by only five mouse clicks, see Figure 5-4. The assembler language syntax is described in the assembler file “LanguageSyntax.asm” by numerous examples. It supports several pseudo operators as well as numerical directives. The debugging dialog offers numerous debugging features like showing the actual 6502 processor register content, single- and multi-instruction debugging, as well as defining and managing break points. Changes in the RAM can be observed in the lower right 6502 Memory Content window, where you can also modify the content of specific RAM addresses. In case of an assembler error, the erroneous source code line will be reported in the Report Window. To show this line in the source code, use the menu item “Show Source Line …”.

The assembler source code in the left upper window can be edited in the usual way with copy, paste, cut and undo commands including select all and find string functionality. A comfortable programming dialog is available to immediately flash the assembled binary code into an 8 KByte EEPROM, see Figure 5-3.

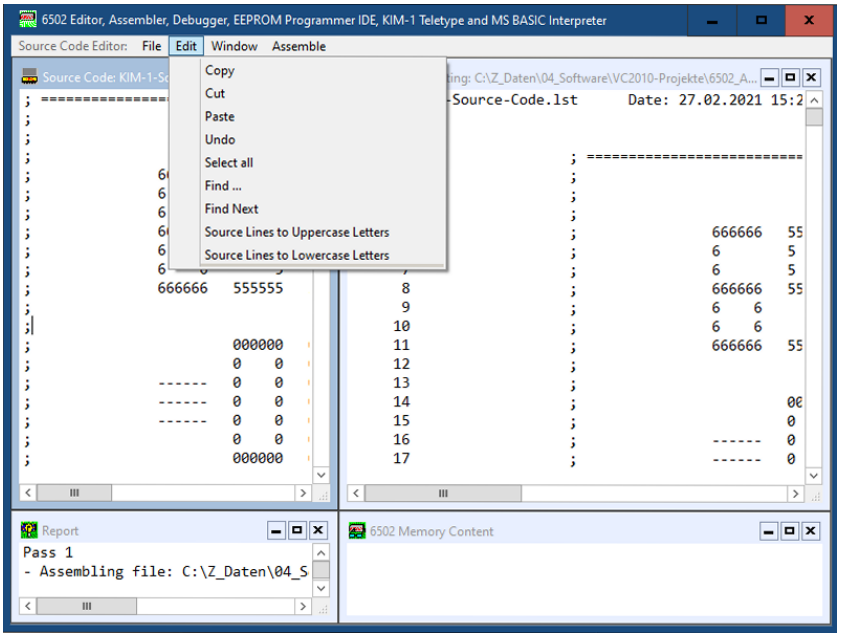
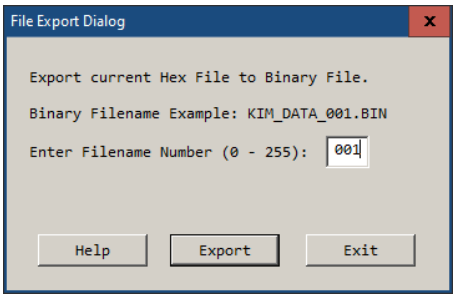


Figure 5-2: IDE Multi Document Interface Application

You can also write assembler programs in the IDE mode, which can be loaded into the RAM area of My-KIM. For this purpose, the IDE provides an export function (in the Assembler Listing Window), which converts the generated Hex file into a binary file, which can be loaded within the Teletype (TTY) mode into the RAM area by the command ‘R’. For details see chapter 5.4.



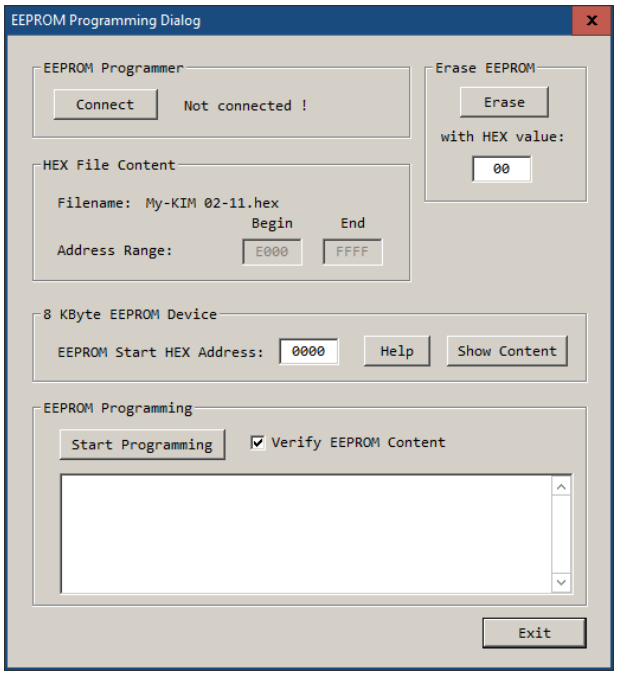


Figure 5-3: EEPROM Programmer Dialog

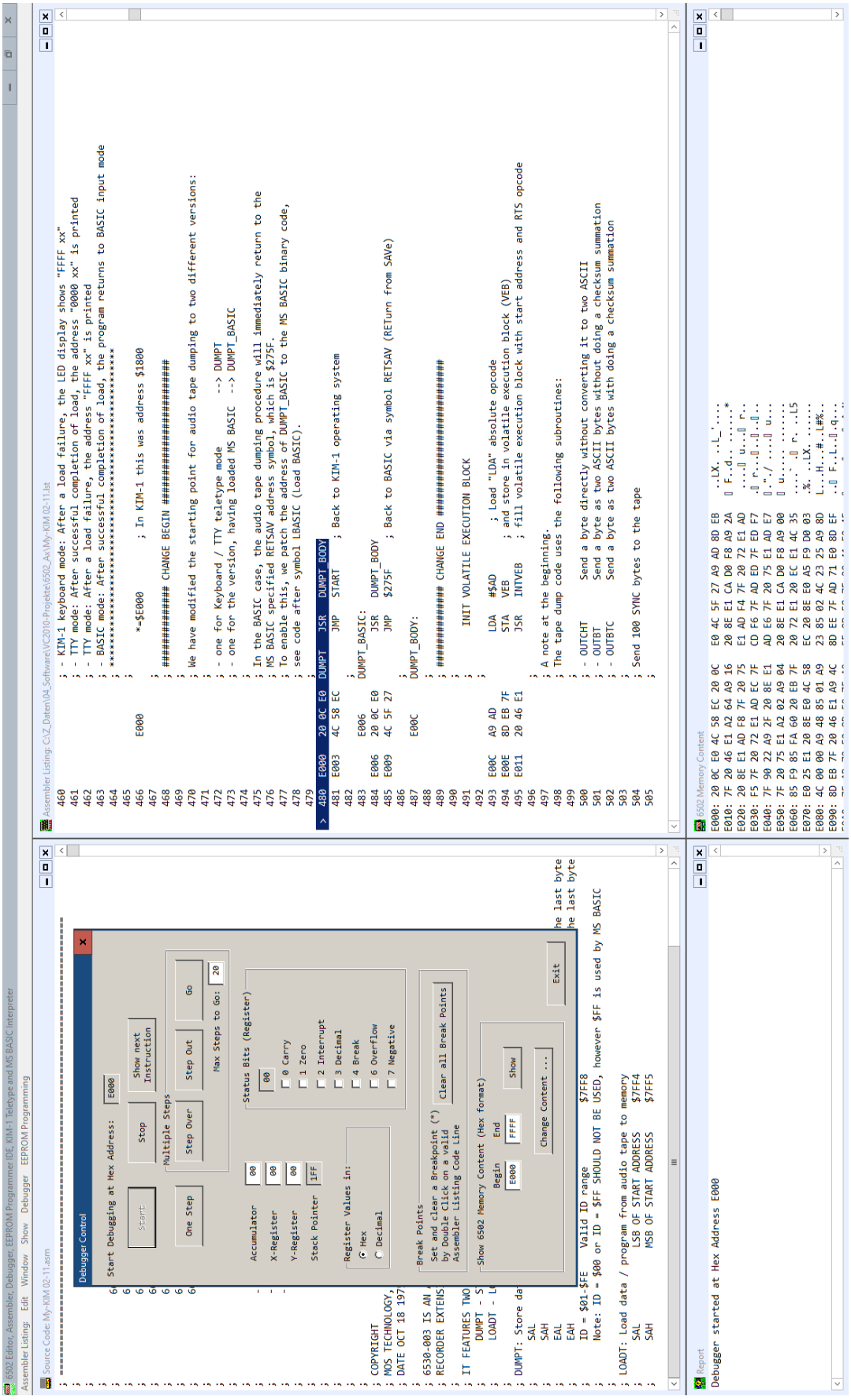


Figure 5-4: IDE with Running Debugging Dialog

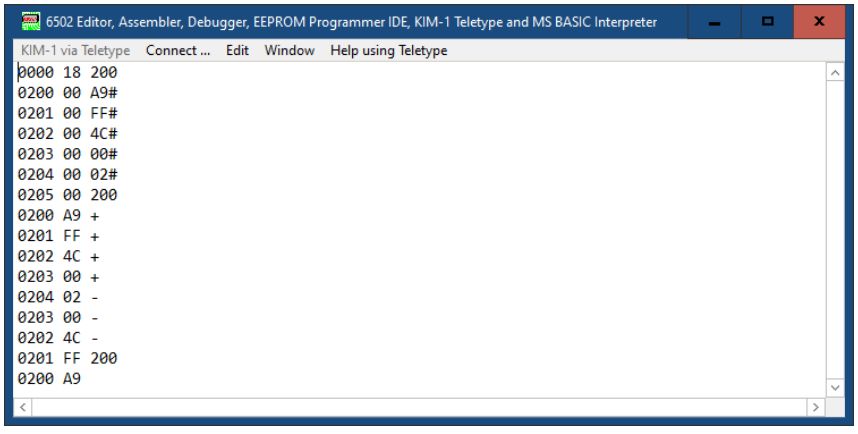
5.4 KIM-1 Teletype Mode

In the KIM-1 Teletype mode the My-KIM computer can be operated from the PC. Before starting this mode, you shall set the mode switch (KB/TTY) to the TTY position and the SST switch to the OFF position. When connecting to KIM-1, the IDE software automatically checks all physically available serial COM lines of the PC, whether the My-KIM computer is logically connected. This means that you don't have to specify a dedicated serial COM port. The RS232 transmission speed is automatically set to 600 Baud, which is equivalent to 60 bytes per second with an 8N1 serial link protocol. To ease the usage in this mode the "Help using Teletype" dialog, which provides all information to successfully operate My-KIM in this mode, see Figure 5-7.

The following examples shows the use of the keys [Space Bar], [#], [+] and [-]:

Display Field	Type the Keys	Description
0000 xx	200 [Space Bar]	Change to address 0200
0200 xx	A9#	Write A9 to address 0200
0201 xx	FF#	Write FF to address 0201
0202 xx	4C#	Write 4C to address 0202
0203 xx	00#	Write 00 to address 0203
0204 xx	02#	Write 02 to address 0204
0205 xx	200 [Space Bar]	Change to address 0200
0200 A9	+	Show next address
0201 FF	+	Show next address
0202 4C	+	Show next address
0203 00	+	Show next address
0204 02	-	Show previous address
0203 00	-	Show previous address
0202 4C	-	Show previous address
0201 FF	200 [Space Bar]	Change to address 0200
0200 A9		

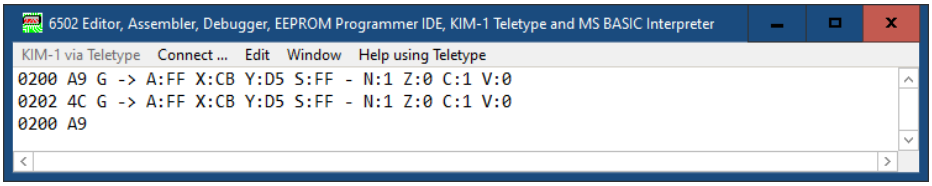
The above procedure is shown below in the Teletype Window:



The program just entered is the following:

Address	OP-Code	Assembler Language Equivalent	
0200	A9 FF	LOOP	LDA #$FF
0202	4C 00 02		JMP LOOP

Now set the SST switch to position ON and push the ‘G’ key to execute the program in the single step mode.



For details on further commands in this mode, use the menu item “Help using Teletype”, see also Figure 5-7.

In this mode programs can be saved or loaded either as an audio tape signal to/from a smartphone App or as a file on the PC. Figure 5-6 shows a recorded audio tape signal clearly showing the recorded FSK signal, as created by the 6502 processor, when writing a user written program with the command ‘W’. Loading an audio tape file is performed by the command ‘L’. The logic, how the audio file is loaded into the memory is described in the “Help using Teletype” dialog.

Warning: When dumping or loading an audio tape file, you shall place the SST switch to position OFF, otherwise it doesn’t work ! This is due to the fact, that the DUMPT and LOADT subroutines use the so-called VEB (Volatile Execution Block) in the RAM area. When the processor reads an instruction from this block in the SST mode, then an NMI interrupt will be performed and the procedure is stopped.

When saving a user written program from My-KIM on the PC with command ‘T’ = Transmit, the program will be saved to two files named “KIM_DATA_XXX.BIN” and “KIM_ADDR_XXX.BIN”, where XXX in the filename represents the three digit decimal number, based on the user defined ID number in memory location $7FF8, e.g. “KIM_DATA_001.BIN”. The ‘DATA’ file includes the user written program based on the start address in $7FF4/5 and the end address +1 in $7FF6/7, while the ‘ADDR’ file only holds two binary bytes, which represent the start address of the first byte in the ‘DATA’ file.

Having pushed the [RS] button, the dump addresses are set to the following default values:

Start Address (Low Byte)	Start Address (High Byte)	End Address (Low Byte)	End Address (High Byte)	Program ID
$7FF4 = $00	$7FF5 = $02	$7FF6 = $80	$7FF7 = $02	$7FF8 = $01

This means the following: When typing the command ‘T’, then the memory area from $0200 to $0280 (128 bytes) will be saved to a file on the PC, having the filename “KIM_DATA_001.BIN”. The result is, that you typically are not required to change these default address values (128 bytes for a program), but only the program ID, if you want to save the program in a file with a different filename. When loading a program with the command ‘R’, the addresses in the memory locations $7FF4 - $7FF7 are ignored and the file on PC with the filename number as defined in $7FF8 will be loaded to the address, which is saved in the file “KIM_ADDR_XXX.BIN”, that mean to the address at the time of saving the program. Saving or loading a program with 128 bytes will take less than 2 seconds.

It shall be noted, that none of the values in the start and end addresses and the ID are modified by the ‘T’ and ‘R’ command.

Warning: When loading a program, never load a program into the zero page area $00EC - $00FF, because there are several My-KIM operating system variables. Same is true for page 1 ($0100 - $01FF), which holds the stack of the operating system and the high RAM memory area $7FE5 -

$7FFF. If you do so, the operating system crashes and does not respond anymore. This is true in the My-KIM stand-alone mode, when loading a program from the smartphone, as well as in the Teletype mode, when loading a program from PC.

Therefore: when writing a program, it is recommended, that you use the memory area from $0200 upwards, which can easily be saved by the command ‘T’ without setting any saving parameters.

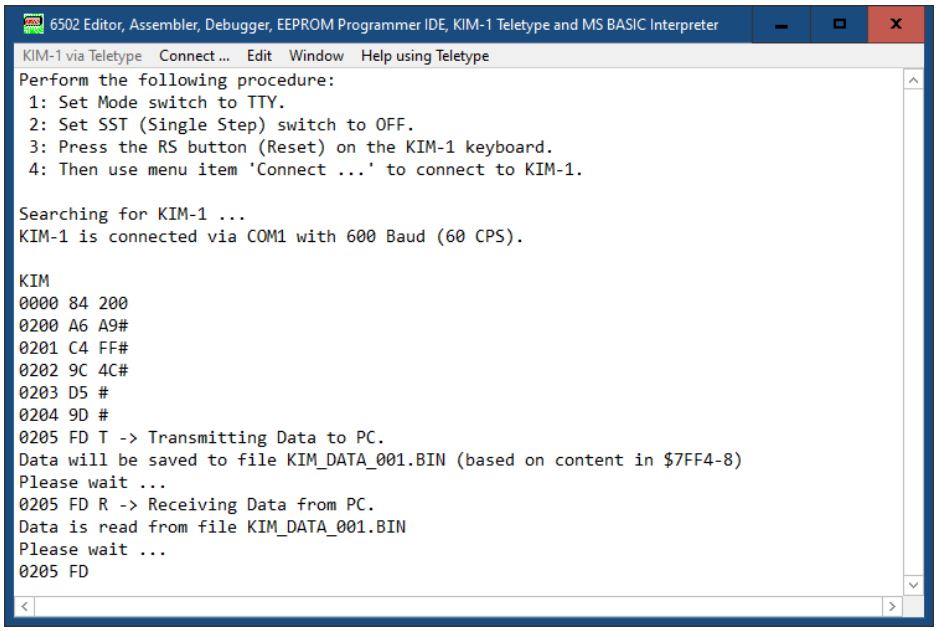


Figure 5-5: My-KIM in the “Teletype” Mode: Transmitting / Receiving a Program from PC

When executing a program with the ‘G’ command (SST switch in OFF position), it is recommended to place a “JMP SHOW” (4C 03 ED) instruction at the end of your program to safely return to the My-KIM operating system. An endless running program can only be stopped by pressing the [ST] button (STOP) on the My-KIM keyboard. As in the Keyboard mode, also in this Teletype mode you can perform single step instructions by setting the SST switch to position ON and then pressing the ‘G’ key.

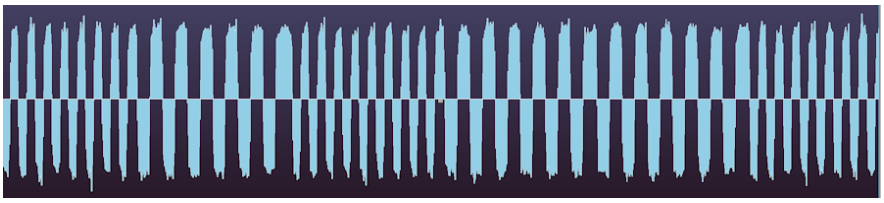


Figure 5-6: Recorded Audio Tape FSK Signal

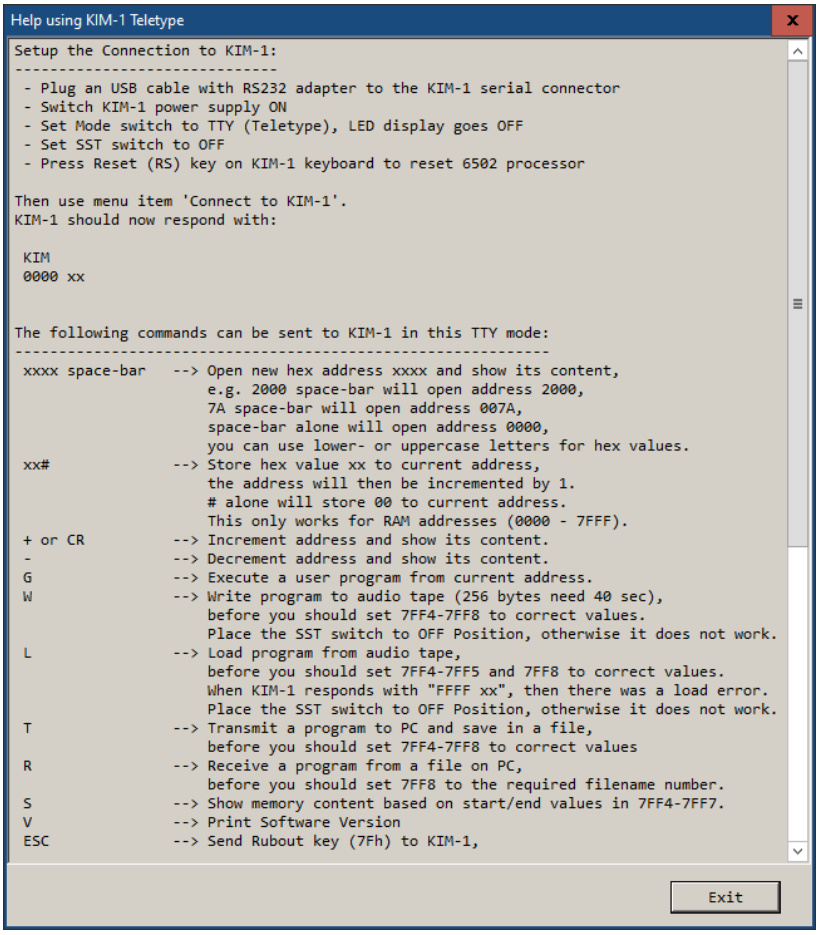


Figure 5-7: Help Information for the “Teletype” Mode

Using the command ‘S’ (Show) will list the memory content, as specified by the entries in $7FF4-$7FF7, which is shown in the Figure 5-8 below. It also shows how to perform single step instructions by the command ‘G’ of a user written program (SST switch to position ON).

When performing single instruction steps, the result of the single step is immediately listed in terms of the content of the accumulator, the X- and Y-Register, the stack pointer and the most important condition flags, like N, Z, C and V Flag.

When the Single Step switch is set to OFF and using the GO command ‘G’, the user defined program starting at the currently shown address will be executed.

To safely return to the My-KIM operating system after having completed your user program, the last operation in your program shall always be a jump to either the symbol “SHOW” at address $ED03 or the symbol “START” at address $EC6B. This is accomplished by writing the op-code “4C 03 ED” or “4C 6B EC” respectively as last instruction to your program. The response of My-KIM is as follows:

Return Symbol / Address		Response	Suggested in
SHOW	$ED03	XXXX yy	Teletype Mode
START	$EC6B	KIM XXXX yy	Teletype Mode
START	$EC6B	XXXX yy	Stand-alone Keyboard Mode

where “XXXX “is the actual content of the address POINTL/H and “yy” is the content of where POINTL/H is pointing to, while POINTL/H has the addresses $00F8/$00F9.

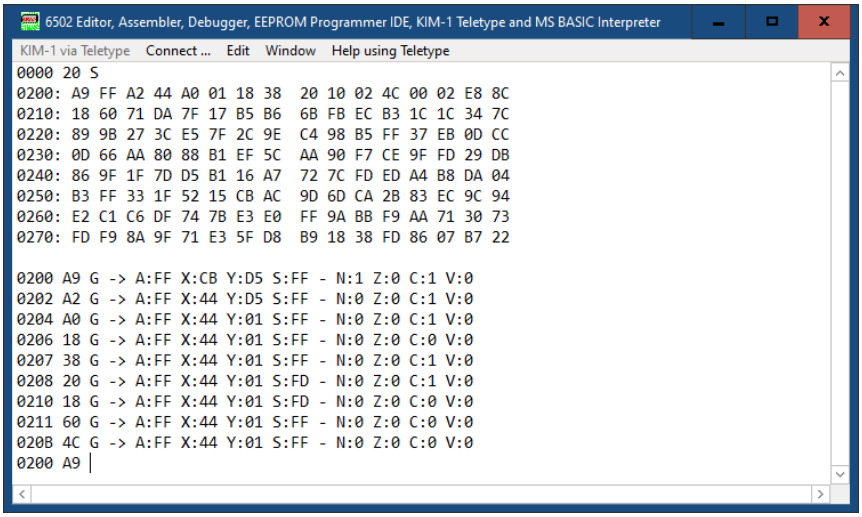


Figure 5-8: Memory Content Listing with Command ‘S’ and Single Step Instructions with Command ‘G’

When you are working in the stand-alone keyboard mode, it is suggested to use the return address “START” = $EC6B, because the SHOW procedure will send status data to the PC, which is not connected in this mode. When working in the Teletype mode, it is suggested to use the return address “SHOW” = $ED03.

5.5 KIM-1 BASIC Mode

In this mode the Microsoft BASIC interpreter will be loaded into the RAM of the My-KIM computer. A prerequisite is, that the file “KB9.BIN” is in the same directory as the started IDE exe file. As in the Teletype mode, My-KIM is automatically connected to a serial COM port. When you type the key ‘M’, Microsoft BASIC is loaded from the binary file “KB9.BIN”, as shown in Figure 5-9. The load procedure takes less than 3 minutes, which is indicated by a progress bar in the edit control window. During the load procedure the window is inactive.

As for the Teletype mode, also in this mode the SST switch shall be in the OFF position, because BASIC is reading instructions from the RAM area, which would generate an NMI interrupt, immediately stopping the BASIC interpreter.

The Figure 5-10 shows the loading of a BASIC program from the default file “BASIC_001.BIN”. Having changed the source line 30, we define by a “POKE 240,2” command, that the program will be saved by the command “SAVE” to the new file “BASIC_002.BIN”.

Having started a BASIC program by the “RUN” command, see Figure 5-11, it can be halted by holding down the Escape key (“Esc”) until BASIC breaks with the message “BREAK IN xx”, where xx is the line number, where BASIC stopped the execution. The program can be resumed by issuing the command “CONT”, as shown in Figure 5-12.

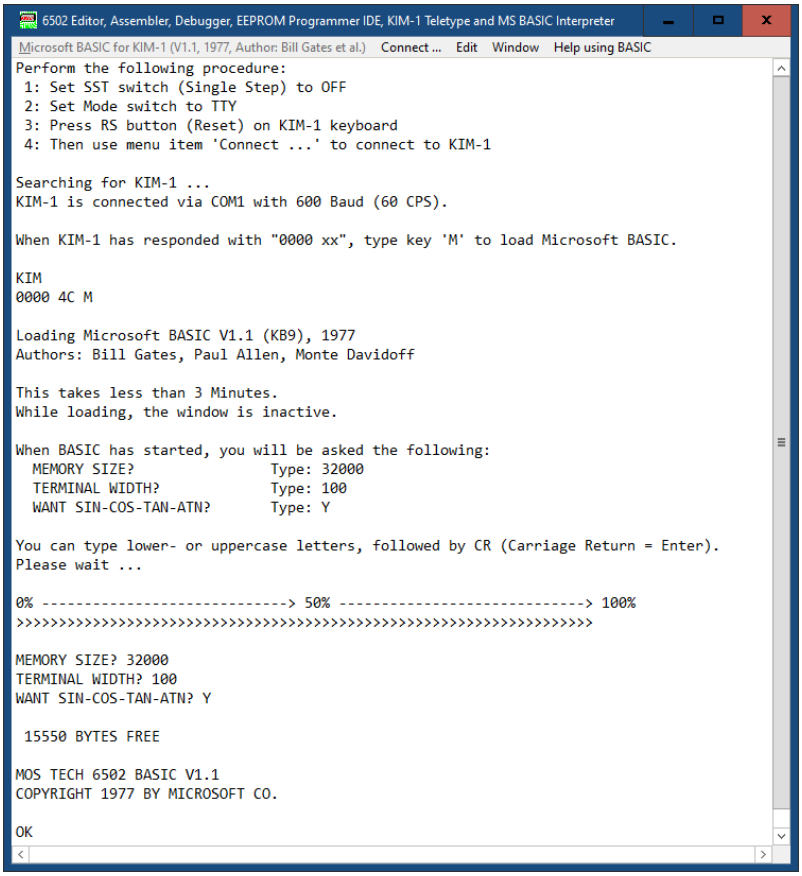


Figure 5-9: Loading Microsoft BASIC into the RAM of the My-KIM Computer

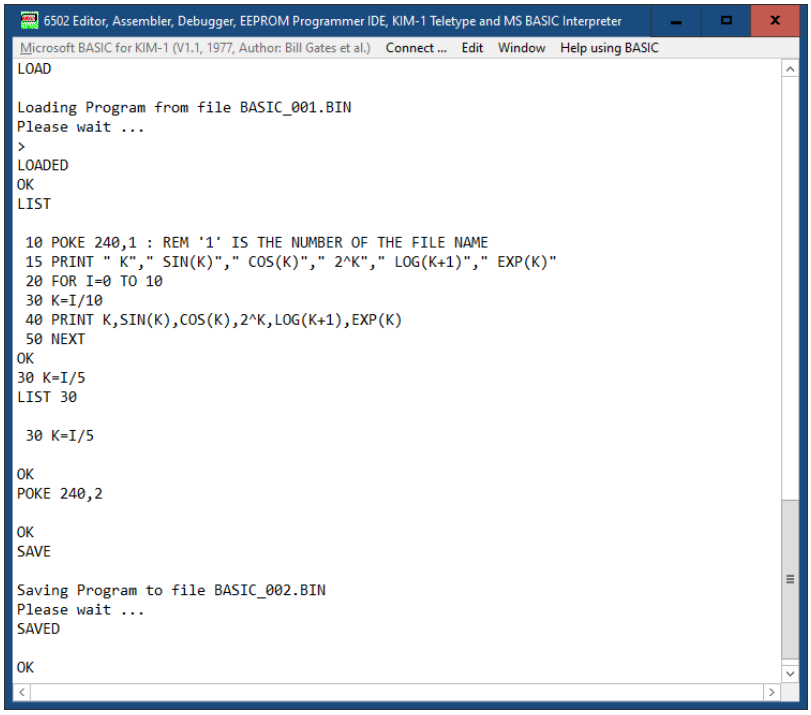


Figure 5-10: Loading a BASIC Program, Editing and Saving in a New File

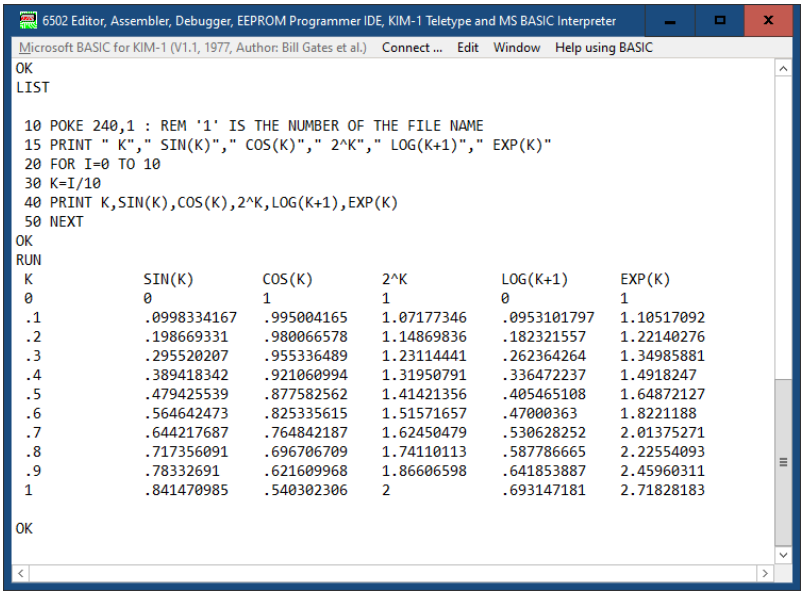


Figure 5-11: Executing a BASIC Program

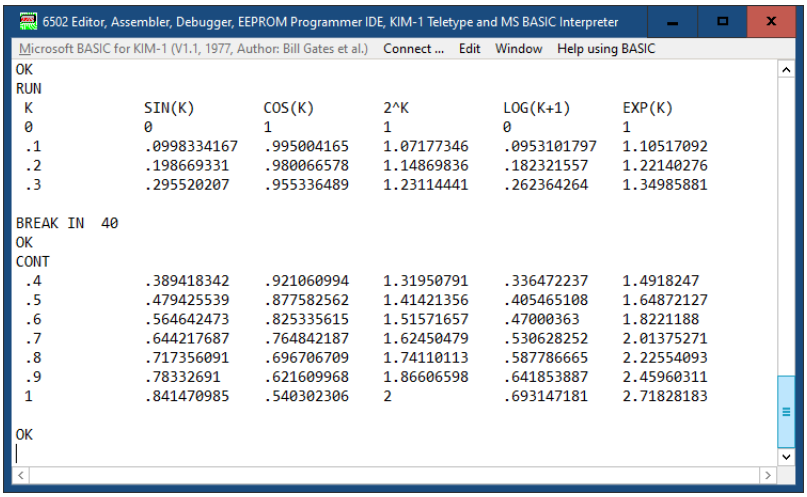


Figure 5-12: Halting and Resuming a BASIC Program (by Holding Down the Escape Key)

Improved Editing of Source Code

The editing of a BASIC source code has been improved with regard to the standard procedure, which is as follows: Having typed in a source code line with a typing error, you are required to type in the complete source code line once more, i.e. editing is not possible.

The new feature implemented in My-KIM is as follows: you can edit a wrong source code line in the BASIC window or in a separate editor, copy it to the clipboard by the menu item “Copy” and finally use the menu item “Insert to BASIC”, which stores the corrected source code line automatically into the BASIC memory of the My-KIM computer. This can be performed for a single source code line as well as for multiple source code lines. When editing a line in the BASIC window, you shall not use the Carriage Return (Enter) key !

This procedure is shown in the following Figure 5-13 to Figure 5-15, where we have completely deleted the source code in the My-KIM computer by the command “NEW” and then inserted the copied source code to BASIC by the above mentioned “Insert to BASIC” procedure.

If you wish to make a print out of your BASIC program, copy the source code listing to the clipboard and insert it into your favourite editor program, from where you can send it to a printer.

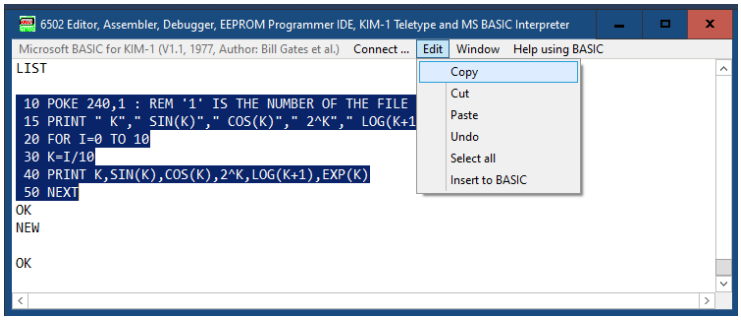


Figure 5-13: Step 1: Selecting and Copying the Source Code

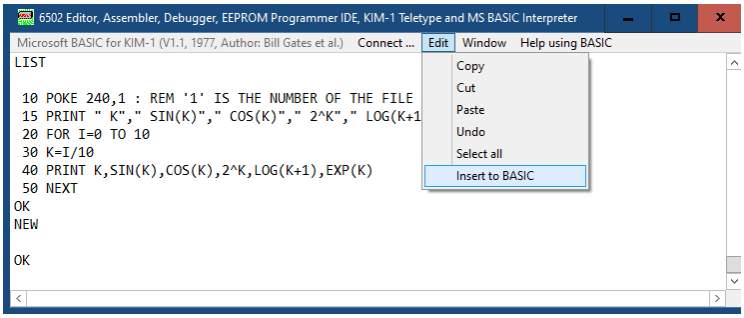


Figure 5-14: Step 2: Inserting the copied Source Code to the BASIC Program

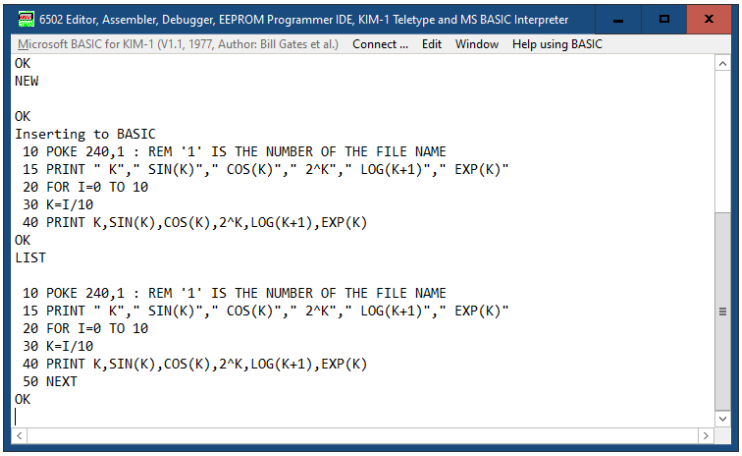


Figure 5-15: Step 3: Final Result

The below shown ‘Help’ dialog shows the most important information how to work with Microsoft BASIC. First steps on using BASIC are described in chapter 7.2, while the BASIC language syntax is summarised in chapter 7.3.

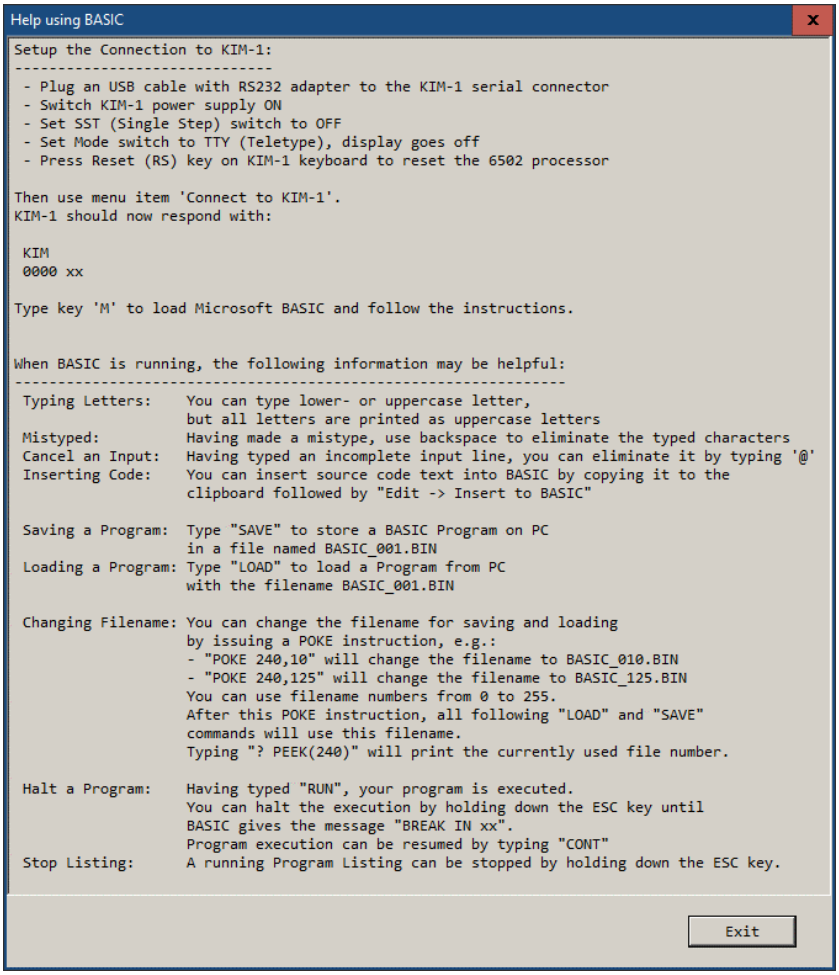


Figure 5-16: Help Information for the Microsoft BASIC Mode

6 Useful My-KIM User Routines

6.1 Audio Tape Dump and Load Routines

A program or a memory range can be dumped or loaded to/from an audio tape in the stand-alone My-KIM keyboard mode. These procedures are described below.

Dump to Audio Tape

To dump a program or the content of a memory range to the audio tape, call the routine DUMPT, which is located at memory address $E000.

To specify the memory range to be dumped to the audio tape, the following memory locations have to be set to the required values:

$7FF4: holds the low byte of the start address
$7FF5: holds the high byte of the start address
$7FF6: holds the low byte of the (end address + 1)
$7FF7: holds the high byte of the (end address + 1)
$7FF8: holds the program ID, which shall be in the range between $01 and $FE

Having pushed the reset button [RS], these addresses hold the following default values:

Start Address:	$0200	($7FF4 = $00, $7FF5 = $02)
End Address:	$0280	($7FF6 = $80, $7FF7 = $02)
ID:	$01	

To start the dumping, set the SST switch to OFF, set the address in the display to “E000” and then start the dump program by pressing the [GO] button. The display will go off. When the routine returns with a display of “0000 xx”, the routine was successful. In case of an error, the display shows “FFFF xx”. The audio tape dumping procedure can be performed in the My-KIM stand-alone mode as well as in the Teletype mode.

Load from Audio Tape

To load a program from the audio tape, call the routine LOADT, which is located at memory address $E073. The associated hex op-code to start the load routine is “4C 73 E0”.

To specify, where the loaded program or data shall be stored in the RAM area of the My-KIM computer, the following memory locations have to be set to the required values:

$7FF4: holds the low byte of the start address
$7FF5: holds the high byte of the start address
$7FF8: holds the program ID

Here the program ID has the following meaning:

ID = $00: Ignore ID and use the Start Address in the Audio Tape Data
ID = $FF: Ignore ID and use the specified Start Address
ID = $01 - $FE: Load Program with exactly this ID and use the Start Address in the Audio Tape Data

The nominal case is to use $00 for the ID without specifying a start address. To start the loading, set the SST switch to OFF, set the address in the display to “E073” and then press the [GO] button, which will switch off the display. When the routine returns with a display of “0000 xx”, the routine was successful. In case of an error, the display shows “FFFF xx”. The audio tape loading procedure can be performed in the My-KIM stand-alone mode as well as in the Teletype mode.

6.2 Audio Tape Calibration Routines

In case of problems in loading the audio tape data, it might be necessary to re-calibrate the receive chain of the audio tape hardware, in particular setting the calibration potentiometer R18 on the Tape / TTY interface board, see Figure 2-5. In this case use the subroutines as defined in the assembler source code “My-KIM_V1.asm” under the heading “DIAGNOSIC PLLCAL TEST FOR NE565”, beginning at label and address PLLCAL = $E269.

6.3 Control Switches and LED Subroutines

The My-KIM computer has two control switches and LEDs. These control elements are also driven by the four non-used I/O pin of the port C 8255 Programmable Peripheral Interface IC (PPI), implemented as two input ports (C4, C5) and two output ports (C2, C3), see Figure 2-8. The output ports are connected to two LEDs and the input ports are connected to two ON/OFF switches (keys). The ports can be set and read by the subroutines in the EEPROM as shown in the following tables. When calling the subroutines to set a port (LED), none of the registers A, X and Y are modified. The subroutines use the Single Bit Set/Reset feature of the 8255 IC.

Table 6-1: Subroutines to Set the Two Output Port Pins (LEDs)

Port	Subroutine	Address	Function	Remark
C2	SPC2_0	$F19E	Set Port C2 to 0 (OFF)	Port C2 is driving the LED 1
C2	SPC2_1	$F1A3	Set Port C2 to 1 (ON)	
C3	SPC3_0	$F1A8	Set Port C3 to 0 (OFF)	Port C3 is driving the LED 2
C3	SPC3_1	$F1AD	Set Port C3 to 1 (ON)	

When calling the subroutines to read the ports (switches), none of the registers X and Y are modified, while the accumulator returns the current status of the read switch.

Table 6-2: Subroutines to Read the Two Input Port Pins (Switches)

Port	Subroutine	Address	Function	Return Value	Remark
C4	RPOC4	$F1B5	Read Port C4	Accumulator returns: 00 hex when switch is OFF 10 hex when switch is ON	Reads status of Switch 1
C5	RPOC5	$F1BB	Read Port C5	Accumulator returns: 00 hex when switch is OFF 20 hex when switch is ON	Reads status of Switch 2

6.4 WAIT Subroutine

For some applications it is necessary to delay the execution of a program sequence. Because My-KIM doesn’t have a timer, this is implemented by a delay loop. The delay loop is called by the subroutine WAIT, which is located at memory location $F0C1. The associated subroutine call hex op-code is “20 C1 F0”. Before you call this subroutine, you have to specify, how long the delay time shall be. This is accomplished by specifying the WAIT_TIME value, which has to be stored to the memory location $7FF9 and is defined as follows (6502 is running with a 1 MHz crystal):

WAIT_TIME = 1 → 13 + 5 + 10 = 23 cycles = 23 micro seconds
WAIT_TIME = 2 → 13 + 10 + 10 = 28 cycles = 28 micro seconds
WAIT_TIME = 3 → 13 + 15 + 10 = 33 cycles = 33 micro seconds
:
WAIT_TIME = 255 → 13 + 1275 + 10 = 1298 cycles = 1298 micro seconds

Based on this we can write:

WAIT_TIME = Delay time in micro seconds / 5.02 – 3.58 with 256 > WAIT_TIME > 0

For realising delay times longer than 1298 micro seconds, use multiple calls of the WAIT subroutine. When calling the WAIT subroutine, none of the registers A, X and Y are modified.

6.5 Memory Test Subroutine 1

If you want to test the RAM memory on correct operation, you can use the subroutine beginning at symbol MT_START, which is located at memory address $F0CB. The associated hex op-code subroutine call is “20 CB F0”. The test program is based on the algorithm, written by Jim Butterfield, published in 1977 in the book “The First Book of KIM”. Before you call this subroutine, you have to specify, which RAM memory area you want to test.

The RAM area to be tested has to be specified by a start page and an end page, where a ‘page’ is defined as 256 bytes. It shall be noted, that the 32 KByte RAM area is between address $0000 and $7FFF. Based on this, the page numbers are defined as:

Page 0 = $00: $0000 - $00FF
Page 1 = $01: $0100 - $01FF
Page 2 = $02: $0200 - $02FF
:
Page 127 = $FF: $7F00 - $7FFF

Set the addresses of the first and last memory page you wish to test to memory locations $0000 and $0001, respectively. Start the program at symbol MT_START. It will stop with a memory address on the display. If no faults were found, the address will be one location past the last address tested. If a fault is found, its address will be displayed.

WARNING:

- Do NOT test the zero page ($0000 - $00FF), because there are our test variables and some KIM-1 variables.
- Do NOT test page 1 ($0100 - $01FF), because there is the KIM-1 stack.
- Do NOT test the last RAM page 127 ($7F00 - $7FFF), because there are some KIM-1 operating system variables.

To test the 32 KByte RAM (excluding page 0, 1 and 127), define the following:

$0000 = $02 (Start = $0200)
$0001 = $7E = page 126 (End = $7EFF)

Then start the memory test program at symbol MT_START = $F0CB and the LED display goes OFF. The test program takes about 8 seconds. The result of the RAM test can be as follows:

- The LED display shows the address “$7F00 xx”, which is the first non-tested RAM address, hence the test was successful.
- If the display shows another RAM address, then the RAM at this address does not work correctly.

It is worth to mention, that this also works in the stand-alone Keyboard and in the TTY mode !

6.6 Memory Test Subroutine 2

There is a second RAM memory test subroutine implemented, which starts at symbol TR = $F134. The associated hex op-code subroutine call is “20 34 F1”. This test subroutine uses the following algorithm:

- Fill the specified RAM range with value $00
- Then set one byte at address TE_AD to $FF (TE_AD = $0201)

- You can define any other address, if necessary for testing, however it should be within the RAM test address range.
- Then check that only this byte has changed in the specified RAM.

WARNING:

- Do NOT test the zero page, because there are our variables and some KIM-1 variables.
- Do NOT test page 1, because there is the KIM-1 stack.
- Do NOT test page 127 ($7E), because there are some KIM-1 operating system variables.

Hence we can check in My-KIM from $0200 to $7EFF (page 2 to page 126).

Example:

Enter to the RAM addresses $0001 and $0002 the RAM page range you want to test, e.g.:

0000 = $02
0001 = $7E

Then start the program at label “TR” = $F134. At the end of the test we can have two situations:

- if NO error occurred, then the LED display shows “$0000 02”
- if an error occurred, then the error address is shown on the LED display.

It is worth to mention, that this also works in the Keyboard and in the TTY mode !

7 Microsoft BASIC Description

7.1 Introduction

Microsoft BASIC was the first commercial project, developed in 1976/7 by Bill Gates, Paul Allen and Monte Davidoff. The first version developed is called KB6 and computes numbers with a precision of 6 digits. The second version, which is used here, is called KB9 and computes numbers with a precision of 9 digits.

The binary code of the KB9 version is stored in the file KB9.BIN. To load BASIC, use the menu item “Start KIM-1 BASIC” in the IDE. BASIC will then be loaded into the memory location, starting at $2000. It has a size of 8815 bytes, i.e. slightly more than 8 K Bytes. Because My-KIM has a RAM size of 32 K Bytes, there are ca. 15550 bytes free for writing a program and storing numerical and string variables. The remaining address range $0200 - $1FFF can be used by the user for other purposes. Starting BASIC will automatically be performed by starting at address 4065 hex. Having loaded BASIC, My-KIM will change some BASIC binary codes to adapt it to the specific environment of My-KIM. One example is, that BASIC can now save a user written program directly to a file on PC, which was not foreseen in the original version, because it could only dump a user program to an audio tape. Loading a user written program from PC is of course also possible. Details on this subject are described in chapter 3.3 and in the My-KIM firmware source code.

BASIC will also use the zero page memory for holding intermediate computational result and housekeeping data in the range $0000-$00DC and at position $00FF. The content of these memory location shall therefore not be changed by the user by e.g. POKE instructions.

When BASIC has been loaded into the My-KIM memory, you are asked the following questions:

Question	Type
MEMORY SIZE?	32000
TERMINAL WIDTH?	100
WANT SIN-COS-TAN-ATAN?	Y

You can type lower- or uppercase letters, but all letters are printed as uppercase ones. Having answered these questions, BASIC will start and prints the available memory space for a program and the variables. For a quick overview on how to use BASIC, open the Help menu item in the IDE, see also Figure 5-16.

7.2 First Steps with BASIC

A note at the beginning:

- all letters typed are automatically converted to uppercase letters (A – Z)
- all commands shall end with a carriage return (CR) key = Enter key
- if you make a typing error, use the backspace key (←) to eliminate the last typed character
- repeated use of the ← key will eliminate previously typed characters
- use the @ key to eliminate the entire line you have typed in

Typing “PRINT 10-4” will result in:

```
6
OK
```

Typing “PRINT 1/2,3*10” will result in:

```
.5          30
OK
```

Instead of typing “PRINT”, you can also use the question mark character like “? 10-4”. The above commands are call “Direct” commands or statements, because the result will immediately evaluated and printed. By using indirect commands, you can type in a program, which can be executed one line after the other. Each indirect command begins with a line number, which can be between 1 and 64000.

Type the following (always with a CR at the end of a line):

```
1 PRINT 2+3
2 PRINT 2-3
```

When typing “LIST”, the following is printed:

```
1 PRINT 2+3
2 PRINT 2-3
OK
```

Now type “RUN” to execute the program, which results in:

```
5
-1
OK
```

When you type “SAVE”, then this small program will be saved on the PC in a file with the filename “BASIC_001.BIN”. When you type “NEW”, then the program will be deleted in the memory of the My-KIM computer. Typing “LOAD” will load the program back from the PC into the My-KIM computer. You can change the filename for saving and loading by using the direct command POKE X,Y. For example, the command:

```
POKE 240,10
```

will change the filename to “BASIC_010.BIN” for all following SAVE and LOAD commands. It shall be noted that no other value than 240 shall be used in this POKE command. If you use another value, it may happen that BASIC crashes !

The complete list of available direct and indirect commands / statements is summarised in chapter 7.3.

7.3 BASIC Syntax

In the following an argument of V or W denotes a numeric variable, X denotes a numeric expression, X$ denotes a string expression and an I or J denotes an expression that is truncated to an integer before the statement is executed. Truncation means that any fractional part of the number is lost, e.g. 3.9 becomes 3.

An expression is a series of variables, operators, function calls and constants which after the operations and function calls are performed using the precedence rules, evaluates to a numeric or string value. A constant is either a number (e.g. 3.14) or a string literal (Zeichenkette).

Table 7-1: Microsoft BASIC Syntax

Statement	Description	Examples
AND	Performs a logical or binary AND operation: Logical AND Example: If expression 1 (A>5) AND expression 2 (B>2) are both true, then branch to line 7 Binary AND Example: 107=01101011, 217=11011001. AND-ing both will result in X=01001001 = 73 = 49 hex	 2 IF A>5 AND B>2 THEN 7 10 X=107 AND 217 → X=73
ASC(X$)	Returns the ASCII numeric value of the first character of the string expression X$. An FC (Function Call) error will occur if X$ is the null string.	10 X$ = “HELLO” 20 I = ASC(X$) → I = 72
ATN(X)	Gives the arctangent of the argument X. The result is returned in radians and ranges from –π/2 to +π/2, where π/2 = 1.5708…	

Statement	Description	Examples
ABS(X)	Gives the absolute values of the expression X.	ABS(1) = 1, ABS(-1) = 1
CHR$(I)	Returns a one character string whose single character is the ASCII equivalent of the value of the argument (I), which must be >= 0 and <= 255.	
CLEAR	Clears all variables, resets FOR and GOSUB state and RESTORE data.	
CONT	Continues program execution after the ESC is typed or after a STOP statement. You cannot continue after any error, after modifying your program, or before your program has been run.	
COS(X)	Gives the cosine of expression X. X is interpreted as being in radians.	
DATA	Specifies data, to be read by a “READ” statement from left to right. Data shall be separated by comma. To restart reading from the beginning, place a “RESTORE” statement. See also “RESTORE” description below.	10 DATA 10.5, 20.7, 30.8 20 READ X : READ Y : READ Z 30 PRINT X,Y,Z 40 RESTORE 50 READ X : READ Y : READ Z 60 PRINT X,Y,Z
DEF	The user can define functions like the built-in functions (SQR, SGN, ABS, etc.) through the use of the DEF statement. The name of the function is “FN” followed by any legal variable name, for example: FNX, FNJ7, FNK0, FNR2. User defined functions are restricted to one line. A function may be defined to be any expression, but may only have one argument. In the example, B and C are variables, that are used in the program. Executing the DEF statement defines the function. User defined functions can be redefined by executing another DEF statement for the same function. User defined string functions are not allowed. “V” is called the dummy variable.	100 DEF FNA(V)=V/B+C 110 Z=FNA(3) Execution of this statement cause Z to be set to 3/B+C, but the value of V would be unchanged
DIM	Allocate space for arrays. All array elements are set to zero by the DIM statement. Arrays can have more than one dimension. Up to 255 dimensions are allowed, but due to the restriction of 72 characters per line the practical maximum is about 34 dimensions. Arrays can be dimensioned dynamically during program execution. If an array is not explicitly dimensioned with a DIM statement, it is assumed to be a single dimensioned matrix of whose single subscript may range from 0 to 10 (eleven elements). If the statement in line 117 is encountered, before a DIM statement for A was found in the program, it would be as if a DIM(10) statement had been executed previous to the execution of line 117. All subscripts start at zero (0), which means that DIM X(100) really allocates 101 matrix elements.	113 DIM A(3), B(19) 114 DIM R3(5,5), D$(2,2,2) 117 A(8)=4
END	Terminates program execution without printing a BREAK message (see STOP). CONT after an END statement causes execution to resume at the statement after the END statement. END can be used anywhere in the program, and is optionally.	
EXP(X)	Gives the constant e=2.71828… raised to the power X. The maximum argument that can be passed to EXP without overflow occurring is 87.3365	X=EXP(2.3) → X=9.97418245
FOR .. NEXT	See example 300: V is set equal to the value of the expression following the equal sign, in this case 1. This value is called the initial value. Then the statements between FOR and NEXT are executed. The final value is the value of the expression following the TO. The step is the value of the expression following STEP. When the NEXT statement is encountered, the step is added to the variable V. If no STEP was specified, it is assumed to be 1. If the step is positive and the new value of the variable V is <= the final value (9.3 in the example) or the step value is negative and the new value of the variable V is >= the final value, then the first statement following the FOR statement is executed. Otherwise the statement following the NEXT statement is executed. All FOR loops execute the statements between FOR and NEXT at least once even the cases like FOR V=1 TO 0. Note that expressions (formulas) may be used for the initial, final and step value of a FOR loop. The values of the expression are computed only once before the body of the FOR…NEXT loop is executed. When the statement after the NEXT is executed, the loop variable (here V) is never equal to the final value, but is equal to whatever value caused the FOR…NEXT loop to terminate. The statements between the FOR…NEXT statements in both examples	300 FOR V=1 TO 9.3 STEP 0.6 : NEXT V (or simply NEXT) 400 FOR V=1 TO 9.3 : NEXT 500 FOR V=10*N TO 3.4/Q STEP SQR(R) 600 FOR V=9 TO 1 STEP -1

Statement	Description	Examples
	(400, 600) would be executed 9 times. Do not use nested FOR loops with the same index variable (here W), as in example 700, which will cause an error. FOR loop nesting is limited only by the available memory.	700 FOR W=1 TO 10 : FOR W=1 TO 20: NEXT W : NEXT W
FRE(X) or FRE(X$)	Gives the number of memory bytes currently unused by BASIC. Memory allocated for STRING memory is not included in the count returned by FRE. The argument passed to the function has no meaning!	270 PRINT FRE(0) or ? FRE(0)
GOTO	Branches to the statement specified by the following line number	130 GOTO 250
GOSUB	Branches to the specified statement until a RETURN is encountered and a branch is then made to a statement after to the GOSUB. GOSUB nesting is limited only by the available memory.	10 I=0 20 GOSUB 100 30 PRINT I 40 END 100 I=I+10 110 RETURN
IF … GOTO	Equivalent to IF … THEN, except that IF … GOTO must be followed by a line number.	150 IF A>5 GOTO 210
IF … THEN	Branches to the specified statement (here line 5) if the relation is true. Executes all of the statement in the remainder of the line 60 after the THEN if the relation (X<0) is true.	50 IF X>10 THEN 5 60 IF X<0 THEN PRINT “X LESS THAN 0”
INPUT	Requests input from the terminal (to be typed in). Each value must be preceded from the previous value by a comma (,). The last value typed should be followed by a carriage return. A ‘?’ is typed as a prompt character. If more data was requested in an INPUT statement than was typed in, a ‘??’ is printed and the rest of the data should be typed in. If more data was typed in than requested, the extra data will be ignored. Strings must be input in the same format as they are specified in DATA statements. This INPUT statement (see line 5) optionally types a prompt string (“VALUE”) before requesting data from the terminal. If carriage return is typed to an input statement, BASIC interrupts and returns to the command mode. Typing CONT after an INPUT command has been interrupted will cause execution to resume at the INPUT statement.	2 INPUT X 5 INPUT “VALUE”; V 7 INPUT “ENTER X”; X,“ENTER Y”;Y 8 INPUT “ENTER STRING”;S$
INT(X)	Returns the largest integer less than or equal to its argument X. The last example will round X to D decimal places.	INT(0.23) = 0, INT(7) = 0 INT(-1) = -1, INT(1.1) = 1 INT(X*10^D+0.5)/10^D
LEFT$(X$, I)	Gives the leftmost I characters of the string expression X$. If I<0 or I>255 an FC error occurs.	? LEFT$(“HELLO”,2) gives “HE”
LEN(X$)	Gives the length of the string expression X$ in characters (Bytes). Non-printing characters and blanks are counted as part of the length.	? LEN(“HELLO”) gives 5
LET	Assigns a value to a variable. “LET” is optionally.	
LIST	Lists current program optionally starting at specified line. List can be stopped by holding down the ESC key. BASIC will then finish listing the current line.	LIST or LIST 100- LIST 100 LIST 100-200
LOAD	Loads a BASIC program from the PC. A NEW command is automatically done before the LOAD command is executed. When done, the LOAD command will print “OK” as a confirmation. The program loaded from PC has the default filename “BASIC_001.BIN”. You can change the number in the filename by issuing a POKE command. For example “POKE 240,10” followed by a LOAD command will load the program with the filename “BASIC_010.BIN”. The user defined number shall be between 0 and 255. You can identify the currently used filename number by issuing the command “? PEEK(240)”.	LOAD → Loads program file “BASIC_001.BIN” POKE 240,130 LOAD → Loads program file “BASIC_130.BIN”
LOG(X)	Gives the natural (base e) logarithm of its argument X. To obtain the base Y logarithm of X use the formula LOG(X)/LOG(Y). Example: the base 10 (common) log of 7 = LOG(7)/LOG(10).	X = LOG(5.6) → X = 1.7227666
MID$(XS, I)	MID$ called with two arguments returns characters from the string expression X$ starting at character position I. If I>LEN(I$), then MID$ returns a null (zero length) string. If I<0 or I>255, an FC error occur.	X$=MID$(“ABCDE”,2) → X$=”BCDE”
MID$(X$,I,J)	MID$ called with three arguments returns a string expression composed of the composed of the characters of the string expression X$ starting at the I-th characters for J characters. If I>LEN(I$), then MID$ returns a null (zero length) string. If I or J <0 or I or J>255, an FC error occur. If J specifies more characters than are left in the string, all characters from the I-th on are returned.	X$=MID$(“ABCDEFG”,3,3) → X$=”CDE”

Statement	Description	Examples
NEW	Deletes current program and all variables.	
NEXT	Marks the end of a FOR loop. If no variable is given, matches the most recent FOR loop. A single NEXT may be used to match multiple FOR statements. NEXT V,W is equivalent to NEXT V : NEXT W	
NOT	Example: IF NOT Q3 THEN 4: If expression “NOT Q3” is true (because Q3 is false), then branch to line 4. Note: NOT -1 = 0 (NOT true = false)	
NULL	Example: “NULL 3”. Sets the number of null (ASCII 0) characters printed after carriage return / line feed. Not needed for electronic terminals.	
ON … GOSUB	Identical to “ON … GOTO” except that a subroutine call (GOSUB) is executed instead of a GOTO. RETURN from the GOSUB branches to the statement after the ON … GOSUB.	
ON … GOTO	The example branches to the line by the I-th number after the GOTO. That is: if I=1 then go to line 10, if I=2 then go to line 20, if I=3 then go to line 30, if I=4 then go to line 40. If I=0 or I attempts to select a non-existing line (I>=5 in this case), the statement after the ON statement is executed. However, if I<0 or I>255 an FC error message will result. As many line numbers will fit on a line can follow an ON … GOTO. The next example statement will branch to line 40 if the expression X is less than zero, to line 50 if it equals zero and to line 60 if it is greater than zero.	100 ON I GOTO 10,20,30,40 105 ON SGN(X)+2 GOTO 40,50,60
OR	Performs a logical or binary OR operation: Logical OR Example: If either expression A<1 or expression B<2 is true, then branch to line 2 Binary OR Example: 107=01101011, 217=11011001. OR-ing both will result in X=11111011 = 251 = FB hex	IF A<1 OR B<2 THEN 2 X=107 OR 217 → X=251
PEEK(I)	The PEEK function returns the contents of the memory address I. The value returned will be between 0 and 255 ($FF). If I>65535 ($FFFF) or I<0, an FC error will occur. An attempt to read a non-existing memory address will return an unknown value, see POKE statement.	J = PEEK(6125) Read the memory location $17ED and write it to the variable J
POKE I,J	The POKE statement stores the byte specified by the argument J into the memory location given by the argument I. The byte to be stored must be between 0 and 255 ($FF) or an FC error will occur. The address I must be between 0 and 65535 ($FFFF) or an FC error will occur. Careless use of the POKE statement will probably cause you to ‘poke’ BASIC to death. That is, the machine will hang and you have to reload BASIC and will lose any program you had typed in. A POKE to a non-existing memory location is harmless. You can use POKE to pass arguments to machine language subroutines, see USR() description.	POKE 6125,10 Write the Byte $0A to memory location $17ED
POS(I)	Gives the current position of the terminal print head (or cursor in window). The leftmost character position on the terminal is position zero and the default right most is 71. If you have entered after loading BASIC “TERMINAL WIDTH?” = 100, the right most position is 99.	
PRINT	Prints the value of expressions on the terminal. Instead of using ‘PRINT’ you can also use the ‘?’ character. If the list of values to be printed out does not end with a comma ‘,’ or a semicolon ‘;’, then a carriage return / line feed is executed after all the values have been printed. Strings in quotes (“) may also be printed. If a semicolon separates two expression in the list, their values are printed next to each other. If a comma appears after an expression in the list, and the print head is at position 56 or more, then a carriage return / line feed is executed. If the print head is before print position 56, then spaces are printed until the carriage is at column 14, 28, 42 or 56. If there is no list of expressions to be printed, then a carriage return / line feed is executed.	10 X=10.5 20 Y=124.7 30 PRINT X 40 ? X 50 PRINT X,Y 60 PRINT X;Y 70 PRINT “X=”;X,”Y=”;Y
REM	Allows the programmer to put comments in his program. REM statements are not executed, but can be branched to. A REM statement is terminated by an end of line, but not by a ’:’.	20 REM THIS IS A COMMENT
RESTORE	Allows to re-reading of DATA statements. After a RESTORE, the next piece of data read will be the first piece listed in the first DATA statement of a program. The second piece of data will be the second piece listed in the first DATA statement, and so on as in a normal READ operation. See examples under “DATA”	
RETURN	Causes a subroutine to return to the statement after the most recently executed GOSUB. See example at GOSUB.	
RIGHT$(X$,I)	Gives the rightmost I characters of the string expression X$. When I<=0 or I>255 an FC error will occur. If I>LEN(X$) then RIGHT$ returns all of	

Statement	Description	Examples
	X$.	
RND(X)	Generates a random number between 0 and 1. The argument X controls the generation of random numbers as follows: X<0 starts a new sequence of random numbers using X. Calling RND with the same X starts the same random number sequence. X=0 gives the last random number generated. X>0 generates a new random number between 0 and 1. The shown example will generate random numbers between 100 and 200.	A=100 B=200 V=(B-A)*RND(1)+A
SAVE	Saves the user typed BASIC program into a file on the PC. The program in memory is unchanged. The program saved to the PC has the default filename "BASIC_001.BIN". You can change the number in the filename by issuing a POKE command. For example "POKE 240,10" followed by a SAVE command will save the program in a file with the filename "BASIC_010.BIN". The user defined number shall be between 0 and 255. You can identify the currently used filename number by issuing the command "? PEEK(240)".	SAVE → Saves to file "BASIC_001.BIN" POKE 240,130 SAVE → Saves to file "BASIC_130.BIN"
SGN(X)	Gives 1 if X>0, 0 if X=0 and -1 if X<0	
SIN(X)	Gives the sine of the expression X. X is interpreted as being in radians.	
SPC(I)	Prints I spaces (or blank) characters on the terminal. Shall be used only in a PRINT statement. X must be between 0 and 255 or an FC error occurs.	
SQR(X)	Gives the square root of the argument X. An FC error will occur if X is less than zero.	
STOP	Causes a program to stop execution and to enter command mode and prints for example "BREAK IN LINE 9000". CONT after a STOP branches to the statement following STOP.	9000 STOP
STR$(X)	Gives a string which is the character expression of the numeric expression X.	10 U$=STR$(3.1) gives U$="3.1"
TAB(I)	Spaces to the specified print position (column) on the terminal. May be used only in PRINT statements. Zero is the leftmost column on the terminal, 71 the most right. If the carriage is beyond position I, then no printing is done. I must be between 0 and 255.	240 PRINT TAB(I)
TAN(X)	Gives the tangent of the expression X. X is interpreted as being in radians.	
USR(I)	Call the user's machine language subroutine with the argument I. See 'USR' function description in chapter 7.8	10 I=5 20 J=USR(I)
VAL(X$)	Returns the string expression X$ converted to a number. If the first non-space character of the string is not a plus '+' or minus '-' sign, a digit or a decimal point '.', then zero will be returned.	10 X=VAL("-3.1") will give X=-3.1
WAIT	The first WAIT example reads the memory (or I/O port) location I, exclusive OR's K with the read memory content, and then AND's the result with J until a non-zero result is obtained. Execution of the program continues at the statement following the WAIT statement. If you are waiting for a bit of a byte to become zero, there should be a '1'-bit in the corresponding position of byte J and K. I must be between 0 and 65535 and J, K must be between 0 and 255. The second WAIT example has only two arguments. Here K is assumed to be zero. Examples: WAIT 6125,1 → Wait until bit 0 of byte in address 6125 (17ED hex) becomes 1 WAIT 3502,64,64 → Wait until bit 6 of byte in address 3502 (0DAE hex) becomes 0	800 WAIT I,J,K 810 WAIT I,J

7.4 Integer Variables

Integer variables can be used in the 9-digit KB9 version of KIM-1 BASIC. Their name must be followed by a % wherever they are used. Note that an integer variable is distinct from a floating point variable of the same name. Integer arrays are also allowed. Integer variables are signed 2 byte values in the range between -32768 to +32767 and hence requires 2 bytes of storage, whereas floating point values require 4 bytes. Non-integer values assigned to an integer variable will be truncated. Integer variables

cannot be used in user defined functions (USR) or “FOR” loops. Integer variables should be used to conserve memory space. They do not save time, in fact, they are usually slower to use than floating point values.

7.5 Floating Point Variables and Memory Space

Simple non-array floating point variables use 6 bytes: 2 bytes for the variable name and 4 for the value. Simple non-array string variables also use 6 bytes: 2 bytes for the variable name and 2 bytes for the length and 2 bytes for the point, where this variable is saved.

Array variables us as minimum 12 bytes: 2 bytes for the variables name, 2 bytes for the size of the array, 2 bytes for the number of dimensions along with 4 bytes for each of the array elements.

7.6 Rules for Evaluating Expressions

Operations of higher precedence are performed before lower precedence operations. This means that multiplications and divisions are performed before additions and subtractions. When operations of equal precedence are found in a formula, the left hand one is executed first.

The precedence of operations used in evaluating expressions is as follows, beginning with the highest precedence: 1) Formulas enclosed in parenthesis; 2) ^ = raise to the power; 3) . = Negation; 4) * / ; 5) + – ; 6) Relational operators like = <> < > >= <= ; 7) NOT; 8) AND; 9) OR.

Relational operator expressions will always have a value of TRUE = -1 or FALSE = 0.

The operators AND, OR and NOT can also be used for bit manipulation and for performing Boolean operations, e.g. “63 AND 16” results in 16 or “4 OR 2” results in 6. These operators convert their arguments to 16 bit signed integer values in the range from -32768 to +32767. If the arguments of these operators are not in this range, an “FC” error results, see Table 7-5.

7.7 Derived Trigonometric Functions

BASIC offers three intrinsic trigonometric functions: SIN(), COS() and ATAN(). Table 7-2 lists the most used trigonometric functions, which can be calculated by existing BASIC functions.

Table 7-2: Derived Trigonometric Functions

Function Name	Function expressed in Terms of intrinsic BASIC Functions
TANGENT	TAN(X) = SIN(X)/COS(X)
INVERSE SINE	ARCSIN(X) = ATN(X/SQR(1-X*X))
INVERSE COSINE	ARCCOS(X) = -ATN(X/SQR(1-X*X))+1.5708
HYPERBOLIC SINE	SINH(X) = (EXP(X)-EXP(-X))/2
HYPERBOLIC COSINE	COSH(X) = (EXP(X)+EXP(-X))/2
INVERSE HYPERBOLIC SINE	ARCSINH(X) = LOG(X+SQR(X*X+1))
INVERSE HYPERBOLIC COSINE	ARCCOSH(X) = LOG(X+SQR(X*X-1))

7.8 BASIC / Machine Language Interface

BASIC allows to perform machine language routines, also call user functions, which are written e.g. by an assembler. A user function can receive an integer value from BASIC and can pass an integer value back to BASIC. There are four steps required to use machine language routines:

1. The KIM-1 version of Microsoft 6502 BASIC starts in memory at address 2000 hex. A machine language routine shall therefore be placed between 0200 hex and 1FFF hex.
2. Store the routine into memory. This can be done either before or after BASIC is loaded:

- Before: By loading a machine program in the TTY mode from the PC
- After: By using POKE instructions

3. Notify BASIC of the location of the user routine by 'USRLOC' which is at address 2040 hex. It must be POKEd to contain the address of the 'USR' machine language routine. 2040 hex (8256 decimal) must be given the low 8 bits of the address and 2041 hex (8257 decimal) must be given the high 8 bits of the address. Invoking the 'USR' function before modifying USRLOC will cause an "ILLEGAL QUANTITY" error since the original content of USRLOC contains the address of the "ILLEGAL QUANTITY" printing error routine.
4. Then the machine language routine has to be called by the 'USR' instruction. BASIC will then jump to the address contained in 'USRLOC'. The 'USR' routine may modify all of the 6502 registers. An RTS op-code (60 hex) should be performed when the routine is completed.

Data can be passed to the machine language routine in two ways:

1. A call (JSR op-code) to the BASIC subroutine (20 C2 F2 = JSR $2FC2) will convert the passed argument of the USR() function to a 16 bit signed integer value in the range of {-32768 to +32767}, which is placed to the zero page address $B1 and $B2. The address $B1 holds the high byte and $B2 holds the low byte of the converted integer value. If the passed argument of the USR() function is outside the range of {-32768 to +32767} an "ILLEGAL QUANTITY" error will result. The original Microsoft documentation states, that the converted value is in the Accumulator and Y-register, but this is not correct and should therefore not be considered. Here Bill made a severe mistake !
2. Data may be written to the unused memory ($0200 - $1FFF) by 'POKE' functions, which can then be processed by the 'USR' routine.

Values may be returned in two ways:

1. A call (JSR op-code) to the BASIC subroutine (20 95 31 = JSR $3195) will cause the 16 bit signed integer in the Y- and A-registers to be returned as the result of the 'USR' function. The Y-register shall hold the low order byte and the Accumulator shall hold the high order byte of the signed 16 bit integer value to be returned to BASIC.
2. The 'USR' routine can store values in memory unused by BASIC, which may be read in by BASIC through 'PEEK' functions.
3. It shall be noted, that the returned value of the USR() user function is only a signed 16 bit integer value.

Based on the above we can summarise the following:

- Using the argument of the 'USR' function we can only transfer a signed integer value to a user routine
- Using the 'POKE' and 'PEEK' functions we can transfer several arbitrary numbers to a user routine

The following table shows an example how to call a user defined machine language routine from BASIC, located at address 0200 hex. The user defined function USR(X) adds 15 to the value passed as argument (X=8120). The USR() routine then returns this value to the BASIC variable Z, which becomes 8135.

Table 7-3: Example of a Machine Language User Function, called by Microsoft BASIC

BASIC Code	Machine Language Code of User Function (Assembler)			
10 REM DEFINE USRLOC AT $0200	0200	20 C2 2F	JSR $2FC2	; Convert argument X, saved in $B1-$B2
20 POKE 8256,0	0203	18	CLC	; Clear Carry to prepare ADC
30 POKE 8257,2	0204	A5 B2	LDA $B2	; Get low order byte of passed integer
40 X=8120	0206	69 0F	ADC #15	; Add 15 to low order byte
50 REM CALL THE USR FUNCTION	0208	A8	TAY	; Transfer Accu to Y (low order byte in Y)
60 Z=USR(X)	0209	A5 B1	LDA $B1	; Get high order byte of passed integer
70 PRINT Z	020B	69 00	ADC #0	; Add 0 to high order byte
	020D	20 95 31	JSR $3195	; Prepare to return the USR() result in (A,Y)
	0210	60	RTS	; Return to BASIC

7.9 Symbols and Special Keys

BASIC uses the symbols and keys as summarised in Table 7-4.

Table 7-4: Symbols and Key

Symbol / Key	Description	Examples
=	Assigns a value to a variable. The 'LET' statement is optional	
.	Negation: "B.A" is subtraction (B-A), while ".A" is negation (-A)	
^	Raise to the power	0^1=1, A^B with A negative and B not an integer gives an FC error
*	Multiplication	
/	Division	
+	Numerical addition and string concatenation. The resulting string must be less than 256 characters in length or an LS error will occur.	10 C1$="WE " : C2$="ARE" 20 C3$=C1$+C2$ → C3$ is the "WE ARE"
-	Subtraction	
= < >	Number and string comparison operators. String comparison is made on the basis of ASCII codes, a character at a time until a difference is found. If during the comparison of two strings, the end of one is reached, the shorter string is considered smaller. Note that "A " is greater than "A" since trailing spaces are significant.	=, <, >, <=, >=, <>
:	A colon is used to separate statements in the same line. Colons may be used in direct or indirect statements. The only limit on the number of statements per line is the line length. It is not possible to place 'GOTO' and 'GOSUB' to the middle of a line.	10 A=1 : B=2
?	Question mark is equivalent to 'PRINT'. Question marks can also be used in indirect statements.	? 2+2 will print 4 "10 ? X" is equivalent to "10 PRINT X"
@	Erases the current line being typed and types a carriage return / line feed.	
Back-Arrow ←	Back arrow erases the last character typed. If no more characters are left in the line, a carriage return / line feed is issued.	
ESC	Holding down the Escape key will halt a running BASIC program or stops the listing of a program. A halted program can be resumed by issuing the command 'CONT'.	

7.10 Error Messages

After an error occurs, BASIC returns to command level mode and types OK. Variable values and the program text remain intact, but the program cannot be continued and all GOSUB and FOR content is lost.

When an error occurs in a direct statement, no line number is printed, e.g.:

Direct Statement: ?"XXXXXXX" ERROR
Indirect Statement: ?"XXXXXXX" ERROR IN YYYY

In both of the above examples “XXXXXXX” will be the error message, while “YYYY” will be the line number where the error occurred for the indirect statement. The following table lists the possible error message (codes) and their description.

Table 7-5: Error Messages

Code	Error Message	Description
BS	Bad Subscript	An attempt was made to reference an array element which is outside the dimension of the array. This error can occur if the wrong number of dimensions are used in a matrix reference. For instance: A(1,1,1)=Z when A has been dimensioned DIM A(2,2).
DD	Double Dimension	After an array was dimensioned, another dimension statement for the same array was encountered. This error often occurs if the array has been given the default dimension 10, because a statement like A(I)=3 is encountered and then later in the program a DIM A(100) is found.
FC	Function Call Error	The parameter passed to a math or string function was out of range. FC errors can occur to: a) a negative array subscript “A(-1)=0”, b) an unreasonable large array subscript >32767, c) LOG-negative or zero argument, d) SQR-negative argument, e) A^B with A negative and B not an integer, f) a call to a ‘USR’ before the address of the machine language subroutine has been patched in, g) calls to MID$, LEFT$, RIGHT$, INT, WAIT, PEEK, POKE, TAB, SPC or ON … GOTO with an improper argument.
ID	Illegal Direct	You cannot use an INPUT or DEF statement as a direct command.
NF	NEXT without FOR	The variable in a NEXT statement corresponds to no previously executed FOR statement
OD	Out of Data	A READ statement was executed but all of the DATA statements in the program have already been read. The program tried to read too much data or insufficient data was included in the program.
OM	Out of Memory	Program too large, too many variables, too many FOR loops, to many GOSUB’s, too complicated expression or any combination of the above.
OV	Overflow	The result of a calculation was too large to be represented in BASIC’s number format. If an underflow occurs, zero is given as the result and execution continues without any error message being printed.
SN	Syntax Error	Missing parenthesis in an expression, illegal character in a line, incorrect punctuation, etc.
RG	RETURN without GOSUB	A RETURN statement was encountered without a previous GOSUB statement being executed.
US	Undefined Statement	An attempt was made by GOTO, GOSUB or THEN to a statement which does not exist.
/0	Division by Zero	
CN	Continue Error	Attempt to continue a program when none exists, an error occurred, or after a new line was typed into a program.
LS	Long String	A string expression was too complex. Break it into two or more shorter ones.
TM	Type Mismatch	The left-hand side of an assignment statement was a numeric variable and the right-hand side was a string, or vice versa, or a function which expected a string argument was given a numeric one or vice versa.
UF	Undefined Function	Reference was made to a user defined function which had never been defined.

8 6502 Instruction Set

8.1 Instruction Set Op-Code Summary

MNEMONIC	INSTRUCTIONS OPERATION		IMMEDIATE OP	IMMEDIATE #	ABSOLUTE OP	ABSOLUTE N	ABSOLUTE #	ZERO PAGE OP	ZERO PAGE N	ZERO PAGE #	ACCUM. OP	ACCUM. N	ACCUM. #	IMPLIED OP	IMPLIED N	IMPLIED #	(IND,X) OP	(IND,X) N	(IND,X) #	(IND),Y OP	(IND),Y N	(IND),Y #	Z.PAGE,X OP	Z.PAGE,X N	Z.PAGE,X #	ABS,X OP	ABS,X N	ABS,X #	ABS,Y OP	ABS,Y N	ABS,Y #	RELATIVE OP	RELATIVE N	RELATIVE #	INDIRECT OP	INDIRECT N	INDIRECT #	Z.PAGE,Y OP	Z.PAGE,Y N	Z.PAGE,Y #	CONDITION CODES N	Z	C	I	D	V
A D C	A + M + C → A	(4)	69	2	6D	4	3	65	3	2							61	6	2	71	5	2	75	4	2	7D	4	3	79	4	3										√	√	√	-	-	√
A N D	A ^ M → A	(1)	29	2	2D	4	3	25	3	2							21	6	2	31	5	2	35	4	2	3D	4	3	39	4	3										√	√	-	-	-	-
A S L	C ← 7·····0 ← 0				0E	6	3	06	5	2	0A	2	1										16	6	2	1E	7	3													√	√	√	-	-	-
B C C	BRANCH ON C=0	(2)																														90	2	2								-	-	-	-	-
B C S	BRANCH ON C=1	(2)																														B0	2	2								-	-	-	-	-
B E Q	BRANCH ON Z=1	(2)																														F0	2	2								-	-	-	-	-
B I T	A ^ M				2C	4	3	24	3	2																															M_7	√	-	-	M_6
B M I	BRANCH ON N=1	(2)																														30	2	2								-	-	-	-	-
B N E	BRANCH ON Z=0	(2)																														D0	2	2								-	-	-	-	-
B P L	BRANCH ON N=0	(2)																														10	2	2								-	-	-	-	-
B R K	(See Fig 1)													00	7	1																										-	-	√	-	-
B V C	BRANCH ON V=0	(2)																														50	2	2								-	-	-	-	-
B V S	BRANCH ON V=1	(2)																														70	2	2								-	-	-	-	-
C L C	0 → C													18	2	1																										-	-	-	0	-	-
C L D	0 → D													D8	2	1																										-	-	-	-	0	-
C L I	0 → I													58	2	1																										-	-	-	0	-	-
C L V	0 → V													B8	2	1																										-	-	-	-	-	0
C M P	A - M	(1)	C9	2	CD	4	3	C5	3	2							C1	6	2	D1	5	2	D5	4	2	DD	4	3	D9	4	3										√	√	√	-	-	-
C P X	X - M		E0	2	EC	4	3	E4	3	2																																√	√	√	-	-	-
C P Y	Y - M		C0	2	CC	4	3	C4	3	2																																√	√	√	-	-	-
D E C	M - 1 → M				CE	6	3	C6	5	2													D6	6	2	DE	7	3													√	√	-	-	-	-
D E X	X - 1 → X													CA	2	1																										√	√	-	-	-	-
D E Y	Y - 1 → Y													88	2	1																										√	√	-	-	-	-
E O R	A ◊ M → A	(1)	49	2	4D	4	3	45	3	2							41	6	2	51	5	2	55	4	2	5D	4	3	59	4	3										√	√	-	-	-	-
I N C	M + 1 → M				EE	6	3	E6	5	2													F6	6	2	FE	7	3													√	√	-	-	-	-
I N X	X + 1 → X													E8	2	1																										√	√	-	-	-	-
I N Y	Y + 1 → Y													C8	2	1																										√	√	-	-	-	-
J M P	JUMP TO NEW LOCATION				4C	3	3																												6C	5	3					-	-	-	-	-	-
J S R	JUMP INTO SUBROUTINE				20	6	3																																			-	-	-	-	-	-
L D A	M → A	(1)	A9	2	AD	4	3	A5	3	2							A1	6	2	B1	5	2	B5	4	2	BD	4	3	B9	4	3										√	√	-	-	-	-

Doc. No.: My-KIM-001
Issue.: 1
Date: 01.04.2021

6502 Instruction Set

MNEMONIC	OPERATION	IMMEDIATE OP	IMMEDIATE N	IMMEDIATE #	ABSOLUTE OP	ABSOLUTE N	ABSOLUTE #	ZERO PAGE OP	ZERO PAGE N	ZERO PAGE #	ACCUM. OP	ACCUM. N	ACCUM. #	IMPLIED OP	IMPLIED N	IMPLIED #	(IND,X) OP	(IND,X) N	(IND,X) #	(IND),Y OP	(IND),Y N	(IND),Y #	Z.PAGE,X OP	Z.PAGE,X N	Z.PAGE,X #	ABS,X OP	ABS,X N	ABS,X #	ABS,Y OP	ABS,Y N	ABS,Y #	RELATIVE OP	RELATIVE N	RELATIVE #	INDIRECT OP	INDIRECT N	INDIRECT #	Z.PAGE,Y OP	Z.PAGE,Y N	Z.PAGE,Y #	CONDITION CODES N	Z	C	I	D	V
L D X	M → X (1)	A2	2	2	AE	4	3	A6	3	2																																			
L D Y	M → Y (1)	A0	2	2	AC	4	3	A4	3	2													B4	4	2	BC	4	3																
L S R	0 → 7·····0 → C				4E	6	3	46	5	2	4A	2	1										56	6	2	5E	7	3																
N O P	NO OPERATION													EA	2	1																												
O R A	A v M → A	09	2	2	0D	4	3	05	3	2							01	6	2	11	5	2	15	4	2	1D	4	3	19	4	3														
P H A	A → Ms, S - 1 → S													48	3	1																												
P H P	P → Ms, S - 1 → S													08	3	1																												
P L A	S + 1 → S, Ms → A													68	4	1																												
P L P	S + 1 → S, Ms → P													28	4	1																												
R O L	C ← 7·····0 ← C				2E	6	3	26	5	2	2A	2	1										36	6	2	3E	7	3																
R O R	C → 7·····0 → C				6E	6	3	66	5	2	6A	2	1										76	6	2	7E	7	3																
R T I	RETURN FROM INTERRUPT													40	6	1																												
R T S	RETURN FROM SUBROUTINE													60	6	1																												
S B C	A - M - C̲ → A (1)(3)	E9	2	2	ED	4	3	E5	3	2							E1	6	2	F1	5	2	F5	4	2	FD	4	3																
S E C	1 → C													38	2	1																												
S E D	1 → D													F8	2	1																												
S E I	1 → I																81	6	2	91	6	2	95	4	2	9D	5	3	99	5	3												
S T A	A → M				8D	4	3	85	3	2																																		
S T X	X → M				8E	4	3	86	3	2																																		
S T Y	Y → M				8C	4	3	84	3	2													94	4	2																			
T A X	A → X													AA	2	1																												
T A Y	A → Y													A8	2	1																												
T S X	S → X													BA	2	1																												
T X A	X → A													8A	2	1																												
T X S	X → S													9A	2	1																												
T Y A	Y → A													98	2	1																												

(1) ADD 1 TO "N" IF PAGE BOUNDARY IS CORSSED
(2) ADD 1 TO "N" IF BRANCH OCCURS TO SAME PAGE
ADD 2 TO "N" IF BRANCH OCCURS TO DIFFERENT PAGE
(3) C̲ = CARRY NOT = BORROW
(4) IF IN DECIMAL MODE, Z FLAG IS INVALID
ACCUMULATOR MUST BE CHECKED FOR ZERO RESULT

X = X REGISTER
Y = Y REGISTER
A = ACCUMULATOR
S = STACK POINTER
M = MEMORY PER EFFECTIVE ADDRESS
Ms = MEMORY PER STACK POINTER

+ ADD
- SUBTRACT
^ AND
v OR
◊ EXCLUSIVE OR
√ MODIFIED

- NOT MODIFIED
M_7 MEMORY BIT 7
M_6 MEMORY BIT 6
N NUMBER OF CYCLES
NUMBER OF BYTES

8.2 Instruction Set Op-Code Matrix

6502 MICROPROCESSOR
INSTRUCTION SET OP CODE MATRIX

MSD \ 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	
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	
2	**JSR** ABS 2 3	**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	
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	
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	
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	
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** (ABS) 3 6	**ADC** ABS 3 4$	**ROR** ABS 3 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	
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	
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		
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	
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*	
C	**CPY** IMM 2 2	**CMP** (IND, X) 2 6			**CPY** ZP 2 3	**CMP** ZP 2 3	**DEC** ZP 2 3		**INY** Implied 1 2	**CMP** IMM 2 2	**DEX** Implied 1 2		**CPY** ABS 3 4	**CMP** ABS 3 4	**DEC** ABS 3 6	
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	
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	
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	

BRK Implied 1 7	-> OP Code -> Addressing Mode -> Instruction Bytes, N = Machine Cycles	$ Add 1 to N if in decimal mode * Add 1 to N if page boundary is crossed ** Add 1 to N if branch occurs to same page Add 2 to N if branch occurs to different page

8.3 ASCII Table

Decimal	Hex	Char.	Decimal	Hex	Char.	Decimal	Hex	Char.	Decimal	Hex	Char.
000	00	NULL	032	20	Space	064	40	@	096	60	`
001	01	SOH	033	21	!	065	41	A	097	61	a
002	02	STX	034	22	“	066	42	B	098	62	b
003	03	ETX	035	23	#	067	43	C	099	63	c
004	04	EOT	036	24	$	068	44	D	100	64	d
005	05	ENQ	037	25	%	069	45	E	101	65	e
006	06	ACK	038	26	&	070	46	F	102	66	f
007	07	BEL	039	27	‘	071	47	G	103	67	g
008	08	BS	040	28	(	072	48	H	104	68	h
009	09	HT	041	29	)	073	49	I	105	69	i
010	0A	LF	042	2A	*	074	4A	J	106	6A	j
011	0B	VF	043	2B	+	075	4B	K	107	6B	k
012	0C	FF	044	2C	,	076	4C	L	108	6C	l
013	0D	CR	045	2D	-	077	4D	M	109	6D	m
014	0E	SO	046	2E	.	078	4E	N	110	6E	n
015	0F	SI	047	2F	/	079	4F	O	111	6F	o
016	10	DLE	048	30	0	080	50	P	112	70	p
017	11	DC1	049	31	1	081	51	Q	113	71	q
018	12	DC2	050	32	2	082	52	R	114	72	r
019	13	DC3	051	33	3	083	53	S	115	73	s
020	14	DC4	052	34	4	084	54	T	116	74	t
021	15	NAK	053	35	5	085	55	U	117	75	u
022	16	SYN	054	36	6	086	56	V	118	76	v
023	17	ETB	055	37	7	087	57	W	119	77	w
024	18	CAN	056	38	8	088	58	X	120	78	x
025	19	EM	057	39	9	089	59	Y	121	79	y
026	1A	SUB	058	3A	:	090	5A	Z	122	7A	z
027	1B	ESC	059	3B	;	091	5B	[	123	7B	{
028	1C	FS	060	3C	<	092	5C	\	124	7C	|
029	1D	GS	061	3D	=	093	5D	]	125	7D	}
030	1E	RS	062	3E	>	094	5E	^	126	7E	~
031	1F	US	063	3F	?	095	5F	_	127	7F	DEL

9 Project Documentation

Beside this document, the project comes with a My-KIM.zip file, holding supporting documents as shown in the table below.

Directory	Filename	Description
6502_IDE	My-KIM_IDE.exe	IDE executable file as created by Visual Studio. There is no need for any installation
	BASIC_001.BIN	Example BASIC source code file
	BASIC_ERR.BIN	BASIC error source code file, which will be loaded, when a requested BASIC program file does not exist. If this file doesn't exist, it will be created
	KB9.BIN	Microsoft BASIC interpreter, V1.1, 1977, binary file
	My-KIM_V1.asm	My-KIM assembler source code to be flashed to the 8 KByte EEPROM
	LanguageSyntax.asm	Assembler Language Syntax Examples, assembler source code file
EAGLE	BaseBoard.brd, BaseBoard.sch	EAGLE schematics and board file for the Base Board
	Display.brd, Display.sch	EAGLE schematics and board file for the Display Board
	Keyboard.brd, Keyboard.sch	EAGLE schematics and board file for the Keyboard Board
	Programmer.brd, Programmer.sch	EAGLE schematics and board file for the EEPROM Programmer Board
	Tape_TTY.brd, Tape_TTY.sch	EAGLE schematics and board file for the Audio Tape and TTY Board
	LED_Switch.brd, LED_Switch.sch	EAGLE schematics and board file for LED and Switch Board
EEPROM Programmer	Several C source code files	C source code files. Having compiled, the hex file has to be flashed to the ATmega8 controller of the EEPROM Programmer Board via Atmel Studio 7
8255 PPI I/O	8255 Intel Programmable Peripheral Interface.pdf	8255 Data Sheet
KIM-1 Documentation	6502 Instruction Set.pdf	6502 Instruction Set Summary
	KIM-1 Hardware Manual.pdf	Original KIM-1 Hardware Manual
	KIM-1 Programming Manual.pdf	Original KIM-1 Programming Manual
	KIM-1 User Manual.pdf	Original KIM-1 User Manual
	KIM-1-Source-Code.asm	Original KIM-1 Assembler source code as in ROM of 6530-002 / 003
	The First Book of KIM.pdf	The First Book of KIM, published in 1977
Microsoft BASIC	KB9.BIN	Microsoft BASIC, V1.1, 1977, KB9, binary file
	Microsoft BASIC KB9 Source-Code.txt	Assembler source code of Microsoft BASIC, as developed on a PDP-10 with MACRO-10 Assembler
My-KIM Documentation	My-KIM.pdf	This document
	My-KIM_V1.asm, My-KIM_V1.hex	My-KIM assembler source code file. In addition the associated Intel Hex file to be flashed to the 8 KByte EEPROM
	LanguageSyntax.asm	6502 Assembler Language Syntax Examples (Test File)

10 Abbreviations

ASCII	American Standard Code for Information Interchange
BASIC	Beginner's All-purpose Symbolic Instruction Code
BSR	Bit Set Reset feature of Intel 8255 PPI
CAD	Computer Aided Design
CPM	Control Program for Microcomputer
DIP	Dual In-line Package
EAGLE	Einfach Anzuwendender Graphischer Layout Editor
EEPROM	Electrically Erasable Programmable Read Only Memory
FSK	Frequency Shift Keying
IDE	Integrated Development Environment
IRQ	Interrupt Request
KB	Keyboard / Keyboard Mode
KIM	Keyboard Input Monitor
MDI	Multiple Document Interface
NMI	Non Maskable Interrupt
PC	Personal Computer
PLL	Phase Locked Loop
PPI	Programmable Peripheral Interface, Intel 8255
RAM	Random Access Memory
RDY	Ready
ROM	Read Only Memory
RS	Reset
RS232	Recommended Standard 232
SMD	Surface Mounted Device
SST	Single Step
ST	Stop
TTY	Teletype / Teletype Mode
UART	Universal Asynchronous Receiver Transmitter
VEB	Volatile Execution Block