

The KIM 1

D. M. de Boer

After the 3 introductory articles about the microprocessor, we are now going to put a few things into practice. Putting it into practice does not, of course, mean that we are going to build our own computer with a soldering iron in hand. Firstly, because building a computer is time-consuming and mind-numbing work, with the added risk that the whole thing won't work when it's finished. Secondly, because there are already well-designed systems on the market which, thanks to large-scale mass production, are fairly cheap and moreover come with a full guarantee.

Building your own computer is really only advisable for people who already have the necessary experience in this field. No, RB is going to start straight away with the fun side of the new hobby . . . programming and using the microcomputer.

Which system?

Countless systems have already been released on the market, which makes it almost impossible for the layman to make a good choice. Some systems are far too expensive relative to their performance, and the enthusiasm of a beginning computer hobbyist is quickly crushed when he discovers that for nearly a thousand guilders all you can do is make a few lights blink.

RB has therefore surveyed the market for you. In doing so we paid particular attention to the price/performance ratio. We soon came across the KIM 1, a well-thought-out system for not too much money. For the time being we will base all our designs on this microcomputer.

What does the KIM 1 have to offer?

The KIM 1 is a complete microcomputer on a single circuit board. This board also carries a keypad and a display. While programming, we can see on this display which code is stored at which address. Once the power supply is connected, the system is completely ready for use. At your disposal is then a generous memory capacity of 1024 bytes (for program and data storage), 15 connections that can each be individually programmed as input or output, a cassette recorder connection, a teletype connection, two interval timers and finally a monitor program and an audio tape program, which are stored in two ROMs.

The monitor program

When the KIM 1 is switched on, the microprocessor will not see any specific instructions, since the contents of the memory are completely random at that point. When we press a key on the little keypad, only the input to which the key is connected will go low, but because the microprocessor isn't looking at that input, nothing further

will happen. The system is going about its own business entirely and appears inaccessible to the outside world. Fortunately the KIM 1 has a so-called monitor program. This program (which is stored in one of the two ROMs) provides the link between the keypad, displays and memory. The reset key on the keypad directs the microprocessor to the start of this program.

From this moment on the displays light up and we are able to type in addresses and information (data). Typed-in addresses appear (in hexadecimal) on the four leftmost 7-segment displays. The contents of that address can now be seen on the two right-hand displays. It seems as though we can look directly into the memory this way and change the memory contents (if desired). In reality everything goes via the microprocessor, which receives its instructions from the monitor program. So we need the monitor program to be able to program the microcomputer; without this program the display stays dark and the microprocessor does not respond to a key being pressed. When we start a self-written program, the link between memory, display and keypad is broken. Only the stop key and the reset key cause a jump back to the monitor program.

With a self-written program we can also give the keypad or the display a different function. With the Melodiant, described elsewhere in this issue, for example, the names of various notes appear on the display, while the keypad is not used and does not respond either. This shows once again how flexible a microprocessor system is.

Apparent connections between switches, lights, displays etc. can be changed without a soldering iron, simply by giving different instructions.

The audio tape program

The board also has a cassette recorder connection. The components needed to turn the tones on the tape into a digital signal are present.

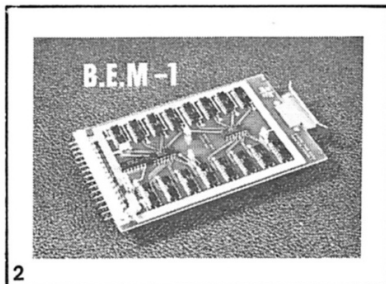
Decoding these digital signals is again taken care of by the microprocessor. So no complicated decoding circuit, just an audio tape program. This program is housed in the second ROM and gives the microprocessor the necessary instructions to place information stored on the tape into the correct memory locations. The audio tape program also contains the necessary instructions to save a program that is in the KIM 1 to cassette. In doing so we must specify which part of the memory is to be recorded. Whether this section contains a program or data (e.g. measurement values) is not important. An identification number must also be given, so that a particular block can be selected later. This cassette recorder connection is an important plus for the hobbyist. He can now build up his own library of programs and can thereby transform the KIM 1 within a few minutes into a Melodiant, calculator, amateur chess player, alarm clock or whatever else he can think of.



The possibilities of the KIM 1

After all this information the reader will be wondering: 'So what can this KIM 1 actually do?' The answer to that question is not easy to give. In the first place it naturally depends on the programmer. It is a fact that many KIM 1

owners are constantly amazed at the possibilities of the KIM 1. Because of the great popularity the KIM 1 already enjoys in America (its popularity is growing in the Netherlands too), a large number of programs have already been written especially for the KIM 1. You have probably heard of a computer that can play chess. We then think of a big cabinet full of blinking lights. It may sound unbelievable, but the KIM 1 manages it too! (see fig. 1). It is very frustrating to lose to a circuit board of less than 22 x 28 cm! Of course you shouldn't expect top-level chess from this microcomputer with its 1k of memory, but a reasonable chess player can count on a good game.



Expansion possibilities of the KIM 1

All address, data and control lines are brought out on the KIM 1's edge connector, which gives us the possibility of expanding the system. This expansion can consist of extra programmable inputs and outputs, extra interval timers, fixed programs in ROMs, or an expansion of the RAM. MOS Technology has developed the KIM 2 . . . 6 specifically for the KIM 1.

The KIM 2 and 3 are RAM expansions (a RAM is a memory that can be both written and read) of 4k and 8k respectively. The KIM 4 is a motherboard used to connect the various KIM boards. The KIM 5 is a ROM expansion board. At present 2 KIM ROMs are available, namely an assembler/editor program and a math package. The assembler/editor program works in combination with a teletype and serves to translate typed-in instructions (such as LDA, STA, INC etc.) into machine codes. We will come back to this in detail at a later stage. Finally, the KIM 6 is an EPROM board with a capacity of 2k bytes. We have not yet discussed the PROM and the EPROM. The PROM is a Programmable Read Only Memory, a memory (ROM) that can be programmed once. So

this memory cannot be written to by the microprocessor. The EPROM (Erasable PROM) is a PROM that can be erased again using UV light.

Other manufacturers also make memory boards that can be connected to the KIM 1. There is, for example, the BEM-1, a 2k RAM board (fig. 2). In the coming months the BEM memory boards will be expanded with the BEM 2 . . . 5. We will keep you informed about developments in this market.

The addressing modes

In the previous installments about the microprocessor we already saw that a complete instruction occupies 1, 2 or 3 bytes. The first byte is the OP code, which indicates what is to be done. The following byte(s) state where the operand can be found in memory. The operand indicates with what something is to be done. There are various ways of indicating where the operand is in memory. These are called the addressing modes.

The 6502 microprocessor, which forms the heart of the KIM 1 microcomputer, has the following addressing modes (for a detailed description see also RB June 1977 p. 216):

Immediate addressing (2 bytes)

The operand is in the 2nd byte of the instruction.

Absolute addressing (3 bytes)

With absolute addressing (also called extended addressing) a full 16-bit address is specified. The operand is at this address.

Zero page addressing (2 bytes)

With zero page addressing (also called direct addressing) the second byte of the instruction gives an address on page 00. The operand is at this address.

Accumulator addressing (1 byte)

All instructions relate to the accumulator (register A). An indication of the operand is therefore not necessary.

Implied addressing (1 byte)

The OP code contains the address of the operand. These instructions relate to the other registers in the microprocessor.

Indexed indirect addressing (2 bytes)

With indexed indirect addressing (abbreviated (IND,X)) the contents of the second byte of the instruction are added to the contents of index register X. The result of this addition gives an address on page 00. At this address the last 8 bits of the final address can be found. The next location on page 00 contains the first 8 bits of the final address.

The operand is at this final address.

Indirect indexed addressing (2 bytes)

With indirect indexed addressing (abbreviated (IND),Y) the second byte of the instruction contains the address of a memory location on page 00. The contents of this memory location are added to the contents of index register Y. The result of this addition gives the last 8 bits of the final address. The first 8 bits of the final address can be found at the next location on page 00. The operand is at the final address.

Indexed zero page addressing (2 bytes)

The contents of the second byte of the instruction are added to the contents of index register X or Y. The result gives an address on page 00. The operand is at this address.

The second and third bytes of the instruction contain a 16-bit base address. The contents of index register X or Y are added to this base address. The operand is at the resulting address.

Relative addressing (2 bytes)

The contents of the second byte of the instruction are added to the contents of the program counter. This addressing mode is used exclusively for jump instructions (branches) in the program.

Absolute indirect addressing (3 bytes)

The contents of the 2nd and 3rd bytes of the instruction give a fully specified memory location. At this memory location are the last 8 bits of the final address. The first 8 bits are at the next memory location. The final address contains the operand.

When you start programming yourself, it is advisable for the time being to use only the first 3 addressing modes. That keeps things clear, and as programs become more complicated, the need for the other addressing modes will arise naturally. The instructions are listed in table 1.

All instructions have standard abbreviations.

When writing a program these abbreviations are always used, because they mean more to us than the numbers we have to type in for an instruction. When the program is finished, we write the machine codes next to the abbreviations. This is also called assembling. Table 1 gives these abbreviations with their meaning.

In a following article we will go further into writing a program yourself. Interested readers are advised to follow the Melodiant program (elsewhere in this issue).

Table 1. The instructions of the 6502 microprocessor

ADC (ADd with Carry) The operand and the carry bit of the status register are added to the contents of register A.

AND (AND) The 'AND' operation is performed between each bit of the operand and register A. The result is placed in register A.

ASL (Arithmetic Shift Left) All bits of the operand shift one place to the left. A '0' is placed in bit 0; bit 7 goes into the carry bit of the status register.

BCC (Branch on Carry Clear) Conditional jump in the program. The jump is made if the carry bit in the status register is zero.

BCS (Branch on Carry Set) Conditional jump in the program. The jump is made if the carry bit in the status register is '1'.

BEQ (Branch on Result Zero) Conditional jump in the program. The jump is made if the zero bit in the status register is '1'.

BNE (Branch on Result not Zero) Conditional jump in the program. The jump is made if the zero bit in the status register is '0'.

BMI (Branch on result Minus) Conditional jump in the program. The jump is made if the negative bit in the status register is '1'.

BPL (Branch on result Plus) Conditional jump in the program. The jump is made if the negative bit in the status register is '0'.

BVC (Branch on V Clear) Conditional jump in the program. The jump is made if the V bit (overflow) in the status register is '0'.

BVS (Branch on V Set) Conditional jump in the program. The jump is made if the V bit (overflow) in the status register is '1'.

BIT (BIT Test) Both the operand and register A keep their original contents. The BIT instruction only causes changes in the carry, negative and overflow bits in the status register.

BRK (BReaK) This instruction generates an interrupt; it is only used when 'checking' a program. We can then stop the program at any desired point.

CLC (CLear Carry) The carry bit in the status register becomes '0'.

CLD (CLear Decimal mode) The add and subtract instructions are performed in binary (normal state).

CLI (CLear Interrupt disable bit) Interrupts are allowed.

CLV (Clear V) The overflow bit in the status register becomes '0'.

CMP (CoMPare) Operand and register A are compared. The status register is set as if the operand were subtracted from register A.

CPX (ComPare with index X) See CMP, except register A becomes index register X.

CPY (ComPare with index Y) See CMP, except register A becomes index register Y.

DEC (DECrement) Decrease the operand by 1.

DEX (DECrement index X) Decrease index register X by 1.

DEY (DECrement index Y) Decrease index register Y by 1.

EOR (Exclusive-OR) The exclusive-OR operation is performed between each bit of the operand and register A. The result is placed in register A.

INC (INCrement) Increase the operand by 1.

INX (INcrement index X) Increase index register X by 1.

INY (INcrement index Y) Increase index register Y by 1.

JMP (JuMP) Jump to a new address.

JSR (Jump to Sub Routine) Jump to a subroutine.

LDA (LoaD register A) Put the operand in register A.

LDX (LoaD index X) Put the operand in index register X.

LDY (Load index Y) Put the operand in index register Y.

LSR (Logic Shift Right) All bits of the operand shift one place to the right. Bit 0 goes into the carry bit of the status register; a '0' is placed in bit 7.

NOP (No OPeration) No instruction is performed.

ORA (OR with register A) The 'OR' operation is performed between each bit of the operand and register A. The result is placed in register A.

PHA (PusH register A) Put the contents of register A on the stack.

PHP (PusH status reg.) Put the status register on the stack.

PLA (PuLL register A) Take the contents of register A from the stack.

PLP (PuLL status reg.) Take the status register from the stack.

ROL (Rotate One bit Left) All bits of the operand shift one place to the left. The carry bit shifts into bit 0, bit 7 shifts into the carry bit.

ROR (Rotate One bit Right) All bits of the operand shift one place to the right. The carry bit shifts into bit 7, bit 0 shifts into the carry bit.

RTI (ReTurn from Interrupt) Continue with the main program.

RTS (ReTurn from Subroutine) Continue with the main program.

SBC (SuBtract with borrow) The operand and C are subtracted from the contents of register A.

SEC (SEt Carry) The carry bit in the status register becomes '1'.

SED (SEt Decimal mode) The add and subtract instructions are performed in binary-coded decimal.

SEI (SEt Interrupt disable bit) Interrupts are ignored.

STA (STore register A) Put the contents of register A in the operand.

STX (STore index X) Put the contents of index register X in the operand.

STY (STore index Y) Put the contents of index register Y in the operand.

TAX (Transfer reg. A to index X) Put the contents of register A in index register X.

TAY (Transfer reg. A to index Y) Put the contents of register A in index register Y.

TXA (Transfer index X to reg. A) Put the contents of index register X in register A.

TYA (Transfer index Y to reg. A) Put the contents of index register Y in register A.

TSX (Transfer Stack pointer to index X) Put the stack pointer in index register X.

TXS (Transfer index X to Stack pointer) Put the contents of index register X in the stack pointer.

The KIM 1 at a reduced price for RB subscribers

The KIM 1 normally retails for f 1110 incl. VAT. Supplied with it are a set of three English-language manuals and an edge connector. Not included is the power supply, for which any device that can deliver 5V at 1.2A and 12V at 0.1A can be used. A diagram of a simple power supply can be found in one of the three manuals.

For the benefit of RB subscribers, the publisher De Muiderkring has taken a batch of KIM 1 microcomputers into stock. These can be supplied at the reduced unit price of f 998 incl. VAT, excl. shipping costs of f 8.15. Readers who are not yet subscribers can also take advantage of this offer by signing up as one now.

A KIM 1 and/or a subscription to RB can be ordered by sending in the 'coupon' below or (if you don't want to damage the magazine) a note stating the relevant details. You can wait with payment until you receive a payment slip (acceptgirokaart). Orders are handled in order of receipt.

Microcomputers ordered via De Muiderkring are guaranteed by the importer under the usual conditions.

COUPON

Cut out or copy onto other paper and send, filled in, in an open, unstamped envelope to: Redactie RB, machtiging 224, 1400VB Bussum.

☐ I am already an RB subscriber

☐ I wish to subscribe to RB (issues Oct., Nov., Dec. '77 and the whole of '78 for only f 40)

☐ I order a KIM 1 microcomputer at the reduced subscriber price of f 998 plus f 8.15 shipping costs

I will transfer the total amount of f immediately upon receipt of your payment slip.

name: _____

address: _____

town: _____

signature: _____