experimenting with the SC/MP (1)

By far the best way to gain a clear understanding of the complexities of microprocessors is through the practical experience of actually constructing and learning to operate one of these 'microcomputers'. For this reason several manufacturers have brought out so-called 'development systems', which are designed to familiarise the user with the particular system in question. By building such a development system oneself, one can not only save considerable expense, but also gain an interesting insight into the field of microprocessor hardware.

H. Huschitt

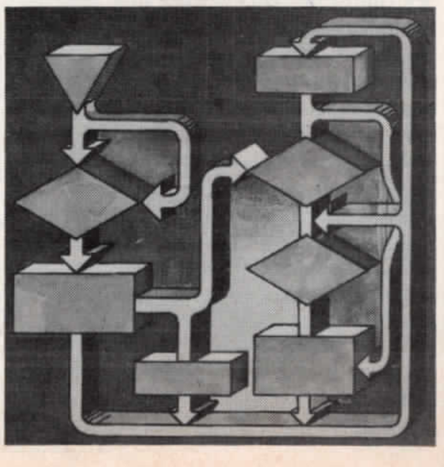
Before commencing, it is perhaps worth reviewing what is 'on the programme' for the next few articles in this series. The flowchart in figure 1 should help the reader to decide if he wants to pursue the topic further. The series calls a temporary halt after part 4 of the series, by which time the budding programmer should possess a microprocessor incorporating a cassette-interface and hexadecimal in- and outputs (using seven-segment displays). For the future, a TV-interface and keyboard are planned, which will convert the development system into a full-grown microcomputer.

SC/MP

SC/MP (pronounced 'scamp') denotes a National Semiconductor microprocessor, type ISP-8A/500D, and stands for Simple Cost-effective Microprocessor. The SC/MP is a modern, low-cost microprocessor. Its design structure or 'architecture' makes it ideally suited to simple applications. Simple two-chip systems (CPU + PROM) can be realised directly using the SC/MP, and assuming that high operating speeds are of secondary importance, the SC/MP can also be used to construct relatively complex systems. There are two versions of the SC/MP available at present. The first and older version (type number ISP-8A/500D) uses P-channel-MOS technology, whilst the more recent SC/MP II (ISP-8A/600D) is an N-MOS version. The two versions are fully compatible, the only difference being that the later version has a higher speed capability, lower power dissipation and requires only one supply voltage.

The SC/MP has 40 pins, of which a number are TTL-compatible. The data- and address buses have tri-state outputs. For most applications, a detailed knowledge of the internal architecture of the IC is not required, and it can be looked upon simply as a 'black box'. The following features however should be noted: the SC/MP has an internal clock generator; this requires only one external component, which can be either a quartz crystal (f_0 = 1 MHz or lower) or a capacitor (C = 500 pF or greater). Since the SC/MP is a static microprocessor, the clock frequency can be lowered as much as desired so that the individual programme steps can be easily distinguished.

Two pulses of the clock oscillator are required for each so-called 'microcycle'. In the case of a 1 MHz crystal each microcycle takes 2 x 1 μs = 2 μs. Depending upon the instruction which it must execute, the basic machine cycle (the combined fetch and execution of a single instruction) requires between 5 and 22 microcycles.



SC/MP registers

The SC/MP has 7 registers which are accessible to the user (programmer), as shown in figure 2.

- *Programme counter (PC, identical to the pointer register Ø)*

The programme counter is a 16-bit register which contains the address of the next programme instruction to be executed by the microprocessor. In order to ensure that all registers in the microprocessor are cleared to zero when the supply is switched on, the NRST (negative reset input) pin must first be set to logic state '0', thereby resetting all the registers. The first '1' or high state will then cause the SC/MP to start up. The first instruction is found under the address ØØØ1 (i.e. not ØØØØ). In programmes with a ØØØØ start address the first instruction must either be a non-memory reference instruction or a NOP instruction (No Operation). Note Ø = Zero, O = letter o.

In order to fetch an instruction from memory, the address, i.e. the content of the PC is put on the address bus. During the NRDS (negative read data strobe) the content of the addressed memory location is put on the data bus, and this instruction byte (byte = 8-bit word) is then delivered to the instruction register and decoded by the instruction decoder. If the MSB (Most Significant Bit) is a '1', then the μP knows that the instruction actually consists of two bytes, in which case, after incrementing the PC, the processor performs a second fetch to obtain the full instruction.

It is worth noting that the PC is always incremented before the instruction is

1

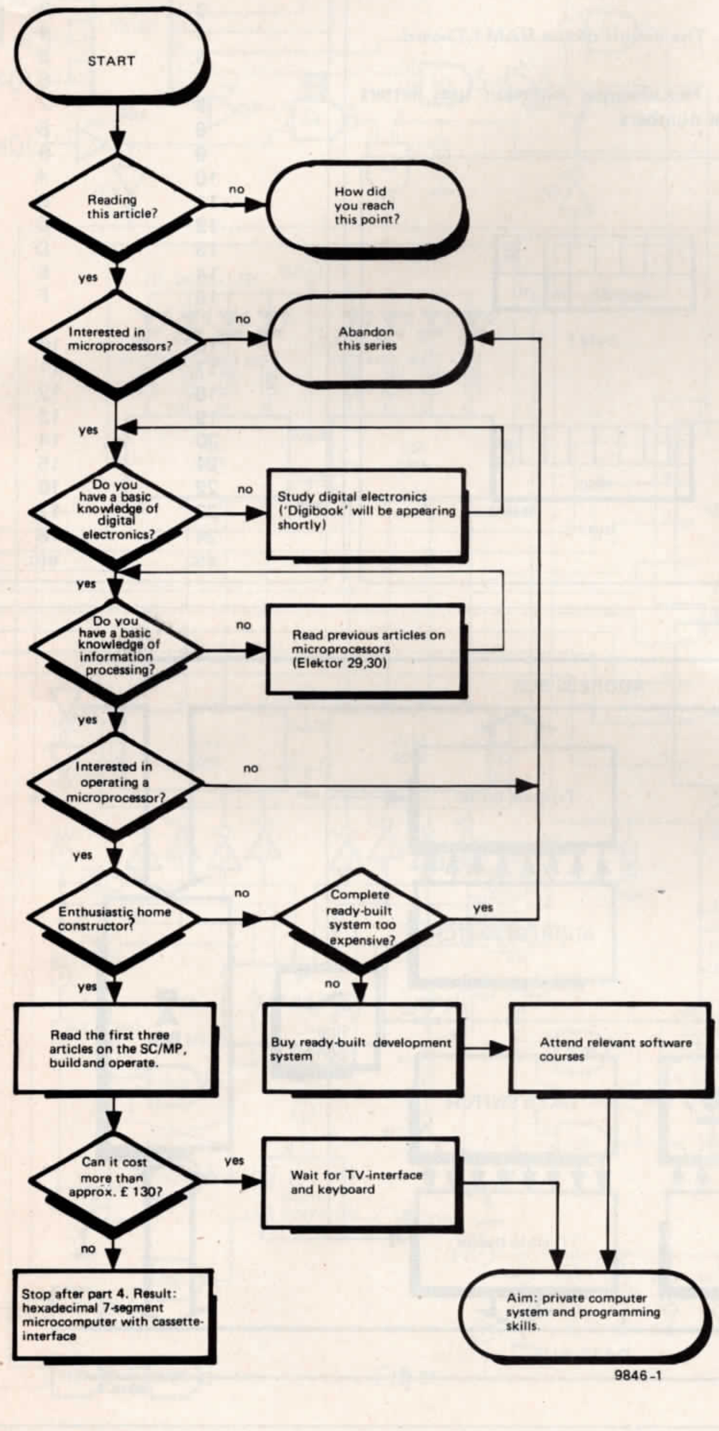

START
Reading this article?
no
How did you reach this point?
yes
Interested in microprocessors?
no
Abandon this series
yes
Do you have a basic knowledge of digital electronics?
no
Study digital electronics ('Digibook' will be appearing shortly)
yes
Do you have a basic knowledge of information processing?
no
Read previous articles on microprocessors (Elektor 29,30)
yes
Interested in operating a microprocessor?
no
yes
Enthusiastic home constructor?
no
Complete ready-built system too expensive?
yes
no
yes
Read the first three articles on the SC/MP, build and operate.
Buy ready-built development system
Attend relevant software courses
Can it cost more than approx. £ 130?
yes
Wait for TV-interface and keyboard
no
Stop after part 4. Result: hexadecimal 7-segment microcomputer with cassette-interface
Aim: private computer system and programming skills.
9846-1

Figure 1. This figure gives an idea of what is 'on the programme' for the SC/MP series.

executed; the only exception to this rule is in the case of jump instructions, when the instruction is executed before the PC is incremented. The actual execution of an instruction requiring arithmetical or logical operations is carried out by the ALU. The whole cycle is then repeated with the next instruction, which is contained in the numerically adjacent address to the first.

● *Pointer-registers (PTR)*

In addition to the programme counter the SC/MP contains three other pointer registers PTR1, PTR2 and PTR3. These are also 16-bit registers used primarily to store addresses. The content of the PC can be exchanged with that of a pointer register so that the programme jumps to the address previously stored in the pointer. The jump back to the main programme will only occur after a second jump instruction (XPPC= Exchange Pointer with Programme Counter). The main programme then picks up where it left off, i.e. at the address which it temporarily stored in the pointer register.

A simple example may help to clarify this manoeuvre. Imagine that Tom (= PC) is hungry and is taking regular bites out of an apple. Dick (= PTR) has had enough, but he's holding a half-eaten pear. At a certain point they are told to swap (XPPC). Tom now has the pear, which he proceeds to demolish with the same regularity originally reserved for the apple. After a second XPPC command Tom find himself again holding the apple and, since Dick wasn't hungry, Tom can continue at exactly the same point that he had reached before the first exchange.

Pointer registers are extremely useful when executing such chores as compiling and reading or storing tables.

● *Accumulator*

The accumulator is an 8-bit register by means of which all manipulation of data is carried out. Only data which are present in the accumulator can be processed by the ALU, and conversely data may only be read into memory via the accumulator. When new data are entered into the accumulator, the data previously held there are lost.

● *Extension Register (E)*

The extension register, which is also an 8-bit register, serves, as its name suggests, as an extension of the accumulator. If information present in the accumulator needs to be retained, then it can be stored in the extension register. In addition, the extension register can be used as a parallel-series or series-parallel converter. To this end it has a series input (SIN, pin 24) and a series output which is buffered by a flip-flop (SOUT, pin 23). An SIO instruction (Serial In/Out) shifts the content of E one bit to the right. The information present at SIN then becomes the highest bit in the extension register and the lowest bit is simultaneously shifted into the buffer flip-flop.

● *Status Register (SR)*

This 8-bit register performs numerous useful functions which will be examined later in the article.

Hexadecimal notation

The only language that a computer understands is its particular 'machine language' or combination of 'noughts' and 'ones'. For example the instruction

2

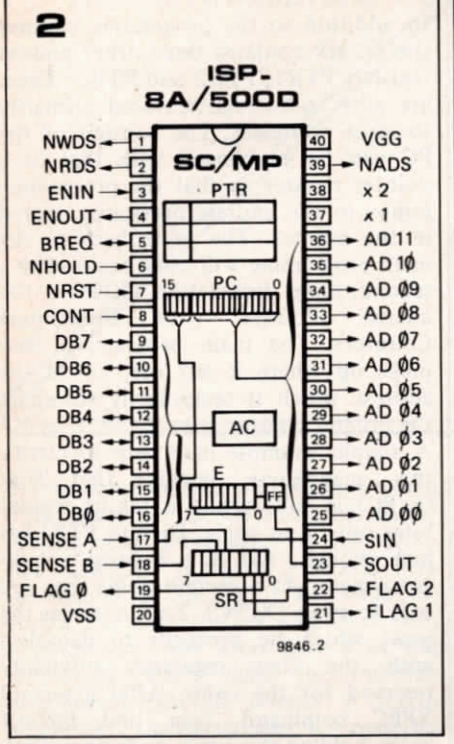


Figure 2. The pin configuration of the SC/MP.

Figure 3. Diagrammatic representation of a two-byte instruction.

Figure 4. Block diagram of the basic version of the SC/MP development system.

Figure 5. The circuit of the RAM I/O-card.

Table 1. Hexadecimal notation uses letters as well as numbers.

3

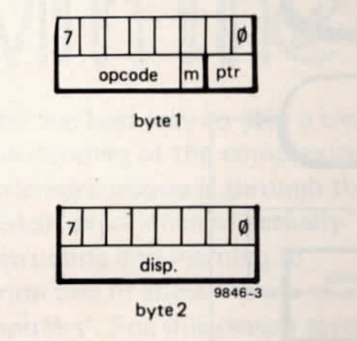
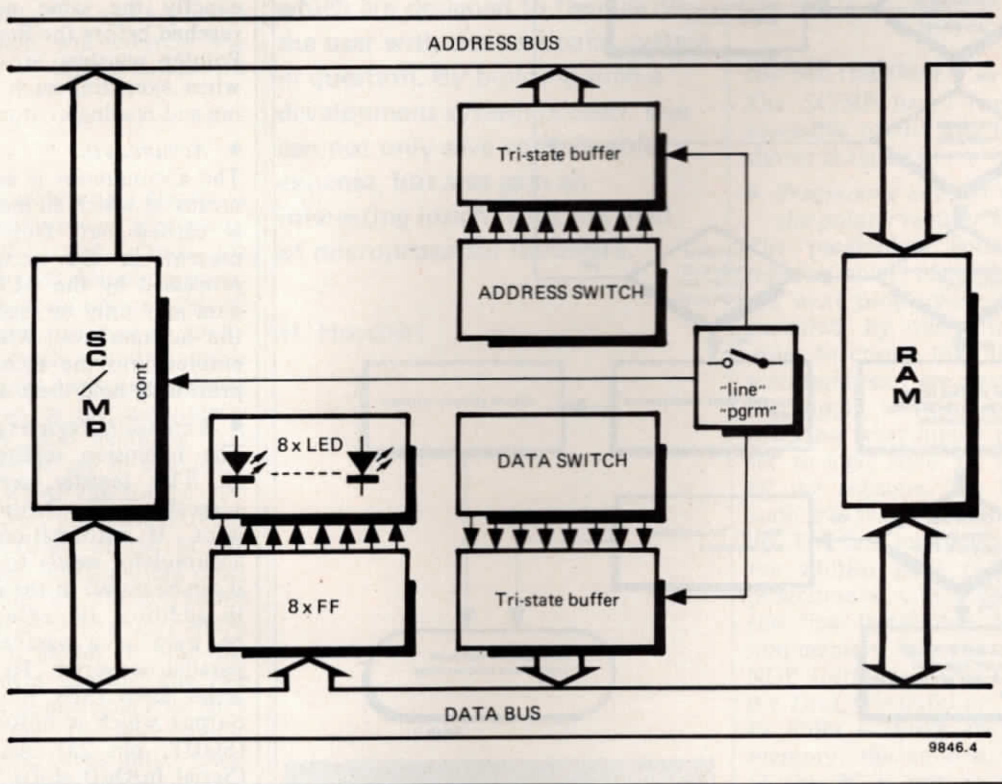


Table 1

decimal	hexadecimal
0	Ø
1	1
2	2
3	3
4	4
5	5
6	6
7	7
8	8
9	9
10	A
11	B
12	C
13	D
14	E
15	F
16	1Ø
17	11
18	12
19	13
20	14
21	15
22	16
23	17
24	18
etc.	etc.

4

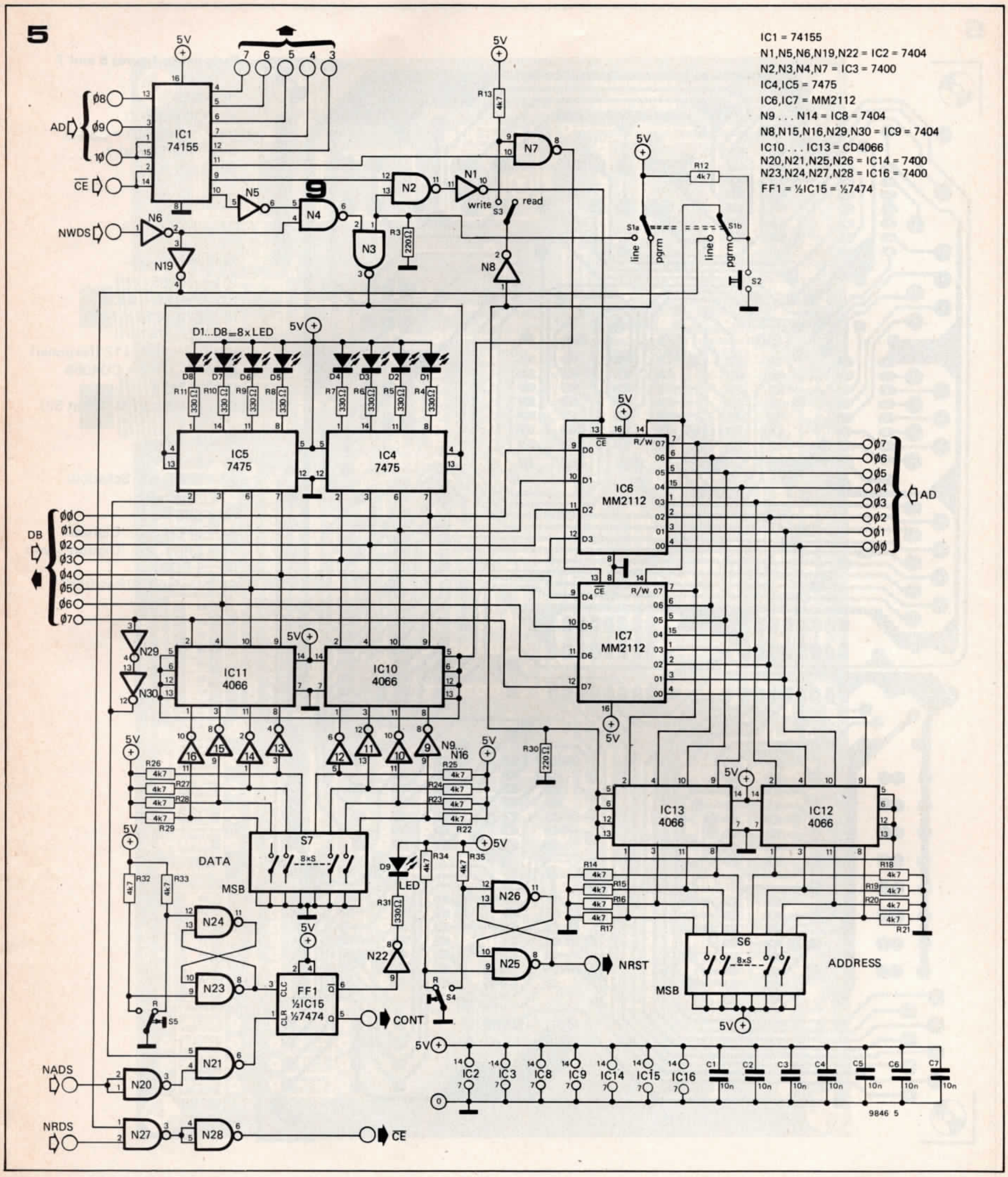


11ØØØ111ØØØØØØØ1 is recognised by the SC/MP as signifying: load the accumulator with the content of the memory location whose address is in pointer register 3, then increment the content of this pointer by 1. It is apparent that when the programmer comes to write out or assemble his programme both the machine language and the longhand translation prove excessively unwieldy. For this reason either so-called assembler language or hexadecimal notation are used.
The above instruction may thus be written either as C7Ø1 or LD @1 (3). The latter is the mnemonic abbreviation or assembler language, used in the SC/MP software, whilst the former is simply the hexadecimal version of the binary code. The assembler language consists of a group of mnemonic letters and symbols (not binary numbers). It can be processed by an assembler programme in the computer, the result being the machine language programme which can then be run in the computer.
The derivation of hexadecimal from binary code was discussed in a previous article (Elektor 30, October 1977), however since working with microprocessors demands a certain proficiency in 'thinking' and being able to calculate in hexadecimal code, the following examples should help to familiarise the prospective programmer with this number system.

Addition: $\begin{array}{r} \text{ØC} \\ \underline{\text{ØC}}+ \\ 18 \end{array}$

Explanation: twelve (C) plus twelve (C) equals twenty-four, which equals one times sixteen plus one times eight (see table 1).

5



Substraction: ØØC3
ØØ1F –
ØØA4

Explanation: fifteen (F) from three does not go, so borrow one (= 16). Fifteen from nineteen is four (4); one was borrowed from C, so that now becomes B. Eleven (B) minus one is ten (A). The result is thus ØØA4.
When subtracting numbers there is the risk that the result will be negative. It is no use placing a minus sign before the offending number since the microprocessor will not recognise it, so in this case the following process occurs. To simplifiy the explanation let us take the example of an up-down counter with decimal outputs which counts down to zero.

counter state	:	ØØØ1
one step down	:	ØØØØ
another step down	:	9999

If instead of decimal the counter displayed hexadecimal, then for the last step shown above, the counter would read FFFF. FFFF is now simply defined as the hexadecimal representation of '–1' or the 'two's complement'* of 1.

By applying the rules of substraction it is also possible to calculate the positive values assigned to represent the negative numbers:

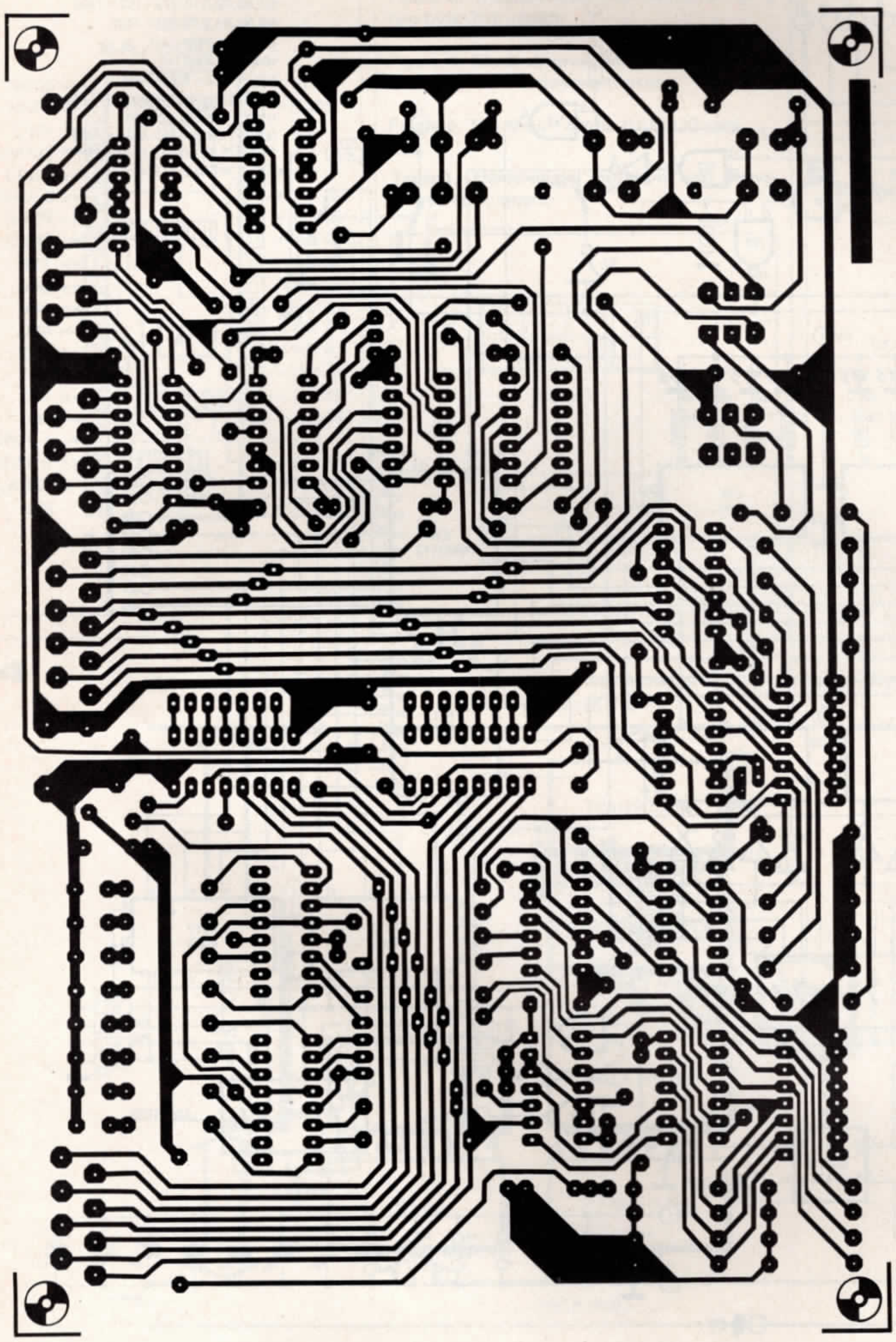
(1)ØØØØ
– ØØØ1
FFFF

(1)ØØØØ
– ØØA4
FF5C

it therefore follows that: –ØØØ1 = FFFF and that –ØØA4 = FF5C.

* The two's complement of a binary number is defined as the number obtained by inverting each bit and adding 1 to the result. For example: the two's complement of ØØØ1 is 111Ø + 1 = 1111.

6



Parts list to figures 5 and 7

Resistors:
R3,R30 = 220 Ω
R4 . . . R11,R31 = 330 Ω
R12 . . . R35 = 4k7
(no R1,R2)

Capacitors:
C1 . . . C7 = 10 n

Semiconductors:
IC1 = 74155
IC2,IC8,IC9 = 7404
IC3,IC14,IC16 = 7400
IC4,IC5 = 7475
IC6,IC7 = MM2112 (National)
IC10 . . . IC13 = CD4066
IC15 = 7474
D1 . . . D9 = LED (D9 in S5)

Miscellaneous:
S1 = DPDT
S2 = SPST, e.g. Schadow 'Digitast'
S3 = SPST
S4 = SPDT, e.g. 'Digitast'
S5 = SPDT, e.g. 'Digitast', with LED
S6,S7 = 8-bit DIP-switch

The highest bit (bit 15) of the numbers designated as negative, is, as the above examples make clear, '1'. The microprocessor recognises this bit as indicating a negative number. In the case of an 8-bit word, capable of representing 256 numbers, the largest possible (positive) number will therefore be Ø1111111 = 7F = 127. That means that, including zero, an 8-bit word may represent 128 positive numbers. The smallest (negative) number possible is, logically enough, 128 − 256 = −128, which equals 1ØØØØØØØ = 8Ø.

Instruction set

The number of possible instructions for the SC/MP is not particularly large, only 46. Although this in no way limits the range of possible applications, it places greater demands upon the amount of storage capacity required and inevitably renders the programme slightly more cumbersome. Thus for extremely large and complex processing chores it is recommended to invest in one of the more sophisticated and expensive types of microprocessor.
A complete description of all 46 instructions would occupy too much space, the reader is therefore referred to the SC/MP data sheet. Some instructions will be examined later in the article.

Instruction format

The SC/MP recognises both one- and two-byte instructions (one byte is 8 bits). The SC/MP instruction guide lists 'operation codes' (opcodes) in hexadecimal form for each instruction.
For some instructions it is necessary to indicate which pointer register is being referred to. This is done by

7

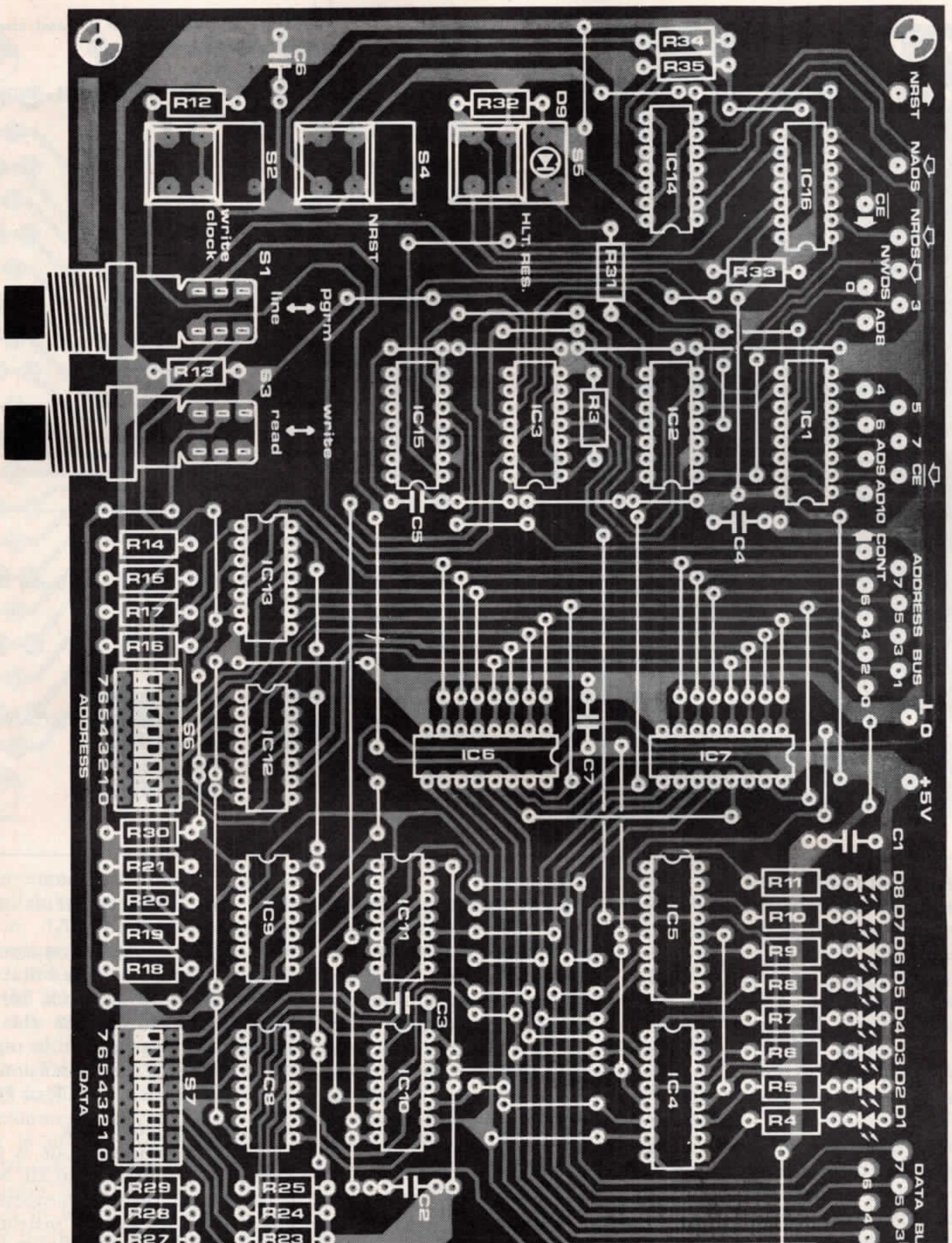


Figure 6. Printed circuit board layout for the RAM I/O-card (EPS 9846-1).

Figure 7. Component layout of the RAM-p.c.b. This board also accomodates all the control switches.

adding the number of the pointer to the opcode. For example, XPPC 3 means 'exchange pointer 3 with PC'. The hexadecimal opcode basis for this instruction is 3C and the instruction refers to pointer 3, so in this case the instruction becomes 3C + 3 = 3F (= ØØ111111).

In the case of two-byte instructions, the opcode is followed by the 'displacement' (see figure 3). For all two-byte instructions this displacement is a number between −128 and +127. The only exception to this rule is the delay instruction (DLY), when the displacement is between Ø and 255. Basically, the displacement gives additional data required for a particular instruction. For instance, if the first byte specifies 'delay', the second byte will specify the duration required; or, if the first byte is a 'load' instruction, the second byte either gives the data or the location where the data are to be found.

Address modes

When data are to be stored in or read out of a specific memory location, it is

8

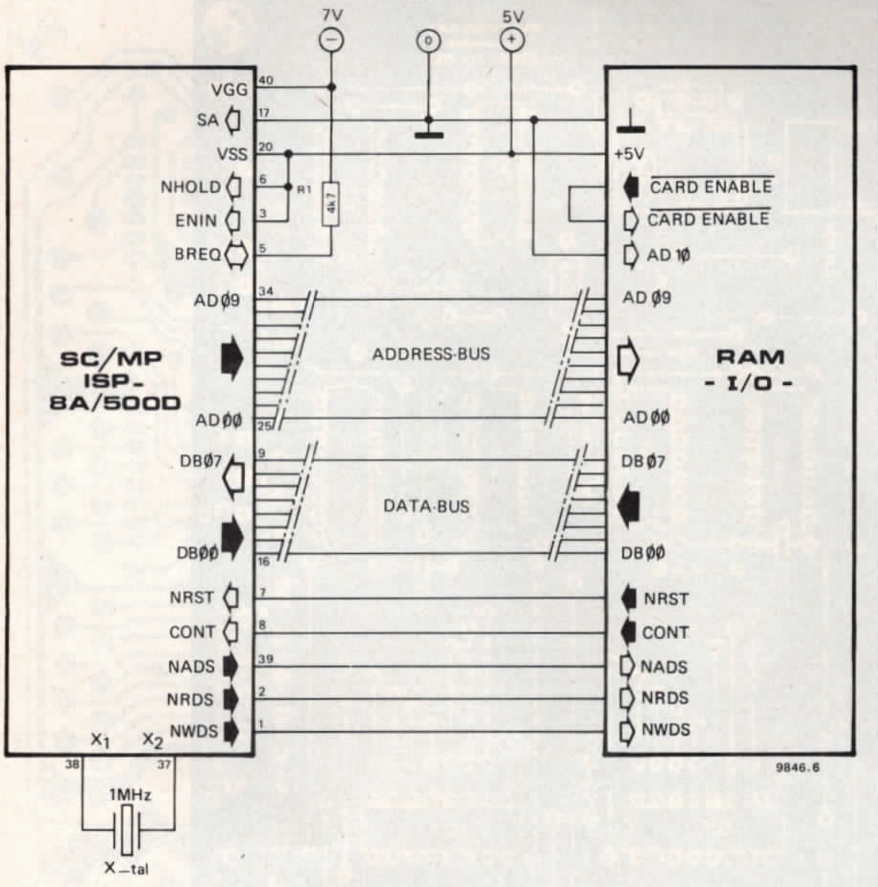


Figure 8. This diagram shows the various connections between the SC/MP and the RAM-card.

Figure 9. The SC/MP printed circuit board (EPS 9846-2).

Figure 10. The component layout of the SC/MP p.c.b.

Parts list to figures 8 and 10.

Resistors:
R1 = 4k7

Semiconductors:
IC = ISP-8A/500D (SC/MP)

Miscellaneous:
1 MHz crystal

clear that the total instruction must contain the address of the location which is being referenced as well as the Read or Write command. Two bytes would be needed to be able to address every memory location of a 64 k memory (= 65,536 locations). Furthermore, one byte is required for the operation code, so that a total of three bytes would be required for the complete instruction.
Several types of microprocessor do in fact use this instruction format. However, the SC/MP adopts a different method requiring only two-byte instructions. This is achieved by using the following address modes:

- *PC-relative addressing*

The content of the PC is used to reference the required address (= effective address = EA). The effective address is obtained by adding the 'displacement' to the content of the PC: EA = (PC) + (disp.). ((PC) signifies 'the content of PC'). Using this method memory locations both 'above' and 'below' the content of the PC can be addressed, since the displacement may be either positive or negative. The highest effective address is logically (PC) + 127, and the lowest (PC) – 128. It is clear that this method does not permit every address of the 64 k memory to be referenced. To achieve this it is necessary to use 'PTR-relative-'or 'indexed addressing'.

- *Indexed addressing*

A 2-byte address is loaded into one of the pointers and the effective address is obtained by adding the displacement to the content of the pointer: EA = (PTR) + (disp). Using this address mode it is possible to reference every location in the 64 k memory, since the pointer may be loaded with any address. Bits Ø and 1 of the instruction byte are used to inform the microprocessor of the number of the pointer in question (see figure 3).

- *Auto-indexed addressing*

This address method is virtually the same as indexed addressing, the only difference being that the content of the pointer is automatically incremented by the value of the displacement.
When the displacement is negative, the pointer content is first altered and then the instruction is executed; when the displacement is positive the instruction is first executed and then the pointer is modified:
neg. disp.: EA = (PTR) + (disp)
pos. disp.: EA = (PTR)
In this address mode bit 2 of the first instruction-byte, the 'modify-bit', is always '1'. This is indicated in the assembler language by the symbol @, which signifies the use of auto-indexed addressing.
In the case of both indexed and auto-indexed addressing, the four highest bits of the pointer remain unchanged since the displacement is a number between –128 and +127.
With regard to these first three address modes it should be noted that when the displacement is –128 (X'8Ø), it is no longer used to obtain the effective address. In this case it is replaced by the content of the extension register: disp. = –128 → EA = (PTR or PC) + (E)!.

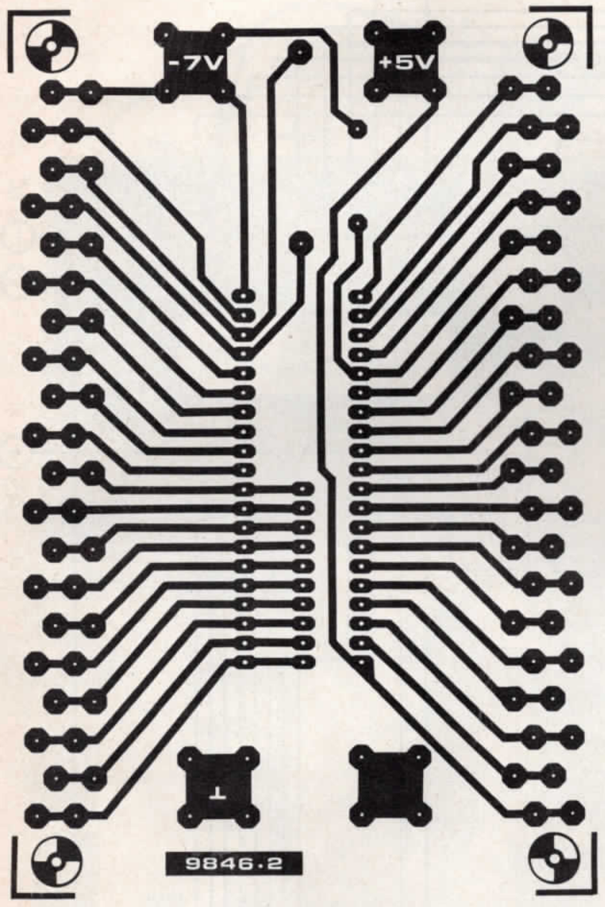
- *Immediate addressing*

This fourth address mode is not really a method of addressing at all! No address is referenced, the microprocessor simply interprets the second instruction-byte as the required data. For example, the instruction LDI X'35 (LDI = Load Immediate) will result in the microprocessor loading the accumulator with X'35 without this number having to be stored somewhere in memory (National Semiconductor use the symbol X' to indicate that what follows is a hexadecimal number).

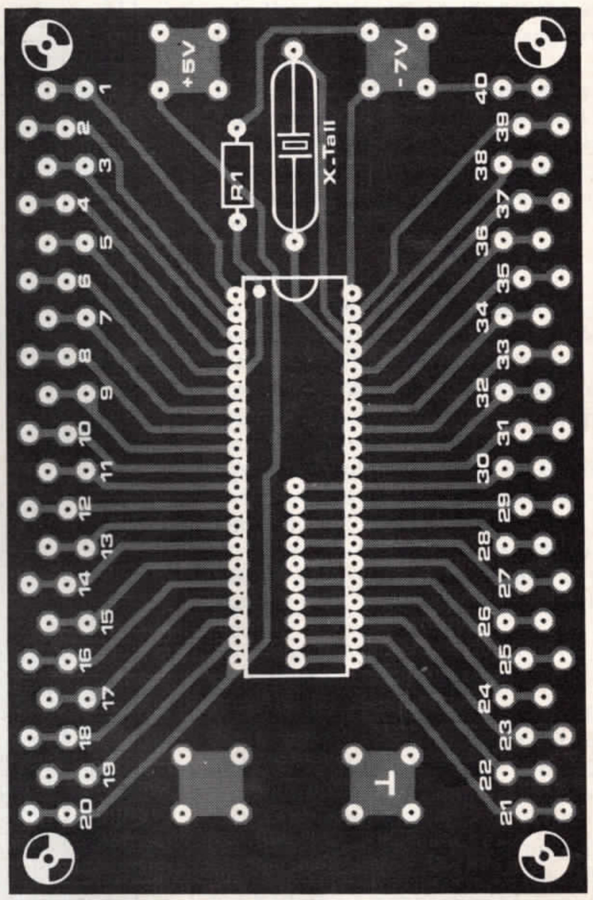
RAM I/O-card

Having hopefully digested the above theory, it is time to get down to practice. The main requirement is a PROM or RAM in which to store the programme for the SC/MP. In order to programme a PROM a special piece of equipment is required, which unfortunately is rather expensive.
A cheaper solution is to use a RAM in conjunction with 'peripheral' hardware

9



10



which will allow a programme to be written in. This is illustrated in the block diagram in figure 4. Both the CPU and the RAM are connected to the address and data buses. Two 8-bit DIP switches are also connected to the data and address buses via tri-state buffer-ICs. These switches form the peripheral hardware which will enable the programme to be written in. When switch S is in the 'pgrm' position, a programme can be loaded into the RAM. The position of the address switch determines in which location the data read in by the data switch will be stored. In this way any desired programme may be written into the RAM. Since an 8-bit switch is used for the addresses, the available storage capacity of the RAM is limited to a 256 x 8 memory.
When switch S is in the 'line' position, the microprocessor will begin to operate and execute the programme stored in the RAM. A visual display is provided in the form of 8 LEDs. During programming, these LEDs will indicate the information present on the data bus. When the programme is running, the LEDs can also be used to display data, providing that they are 'addressed'.

The circuit

Figure 5 shows the complete 'hardware' for the SC/MP memory and peripherals. The 256 x 8 bit RAM is formed by two MM2112 ICs (IC6, IC7). Addresses are gated via buffers IC12 and IC13 onto the parallel-connected address inputs by means of switch S6. Since tri-state buffers are still fairly expensive, 'normal' analogue switches, type CD 4066, are used instead. In the same way, data are gated onto the data bus lines via data switch S7; these data are displayed visually by means of LEDs D1 . . . D8. Since these LEDs are driven by TTL-ICs (IC4, IC5), the inputs of the integrated analogue switches (IC10, IC11) must also be driven by TTL-outputs, hence the need for the inverters N9 . . . N16.
The address-bits AD⌀8 . . . AD1⌀ are connected to IC1 which functions as an address decoder. These bits determine which part of the system is addressed. For the time being, of the 8 available outputs only 3 are used, namely the ⌀, 1 and 2 outputs which address the RAM, the LEDs and the data switch (DS) respectively (see table 2). The remaining decoder outputs can be implemented at a later stage to address additional memory or peripherals. The system is controlled by means of five switches:

- *S1a-S1b, the line-programme switch*

In the 'line' position the SC/MP assumes command of the data- and address buses, whilst S6 and S7 are disabled; in the 'pgrm' position a programme may be loaded into the RAM, by means of S6 and S7.

- *S2, clock-write switch*

By operating this switch, the information represented by the position of S7 is written into the RAM; S2 will only function with S1 in the 'pgrm' position.

- *S3, read-write switch*

The content of the RAM can be checked by setting S3 in the 'read' position. The LEDs display the content of the memory location addressed by S6. S3 will only function if S1 is switched to 'pgrm'.

- *S4, NRST switch*

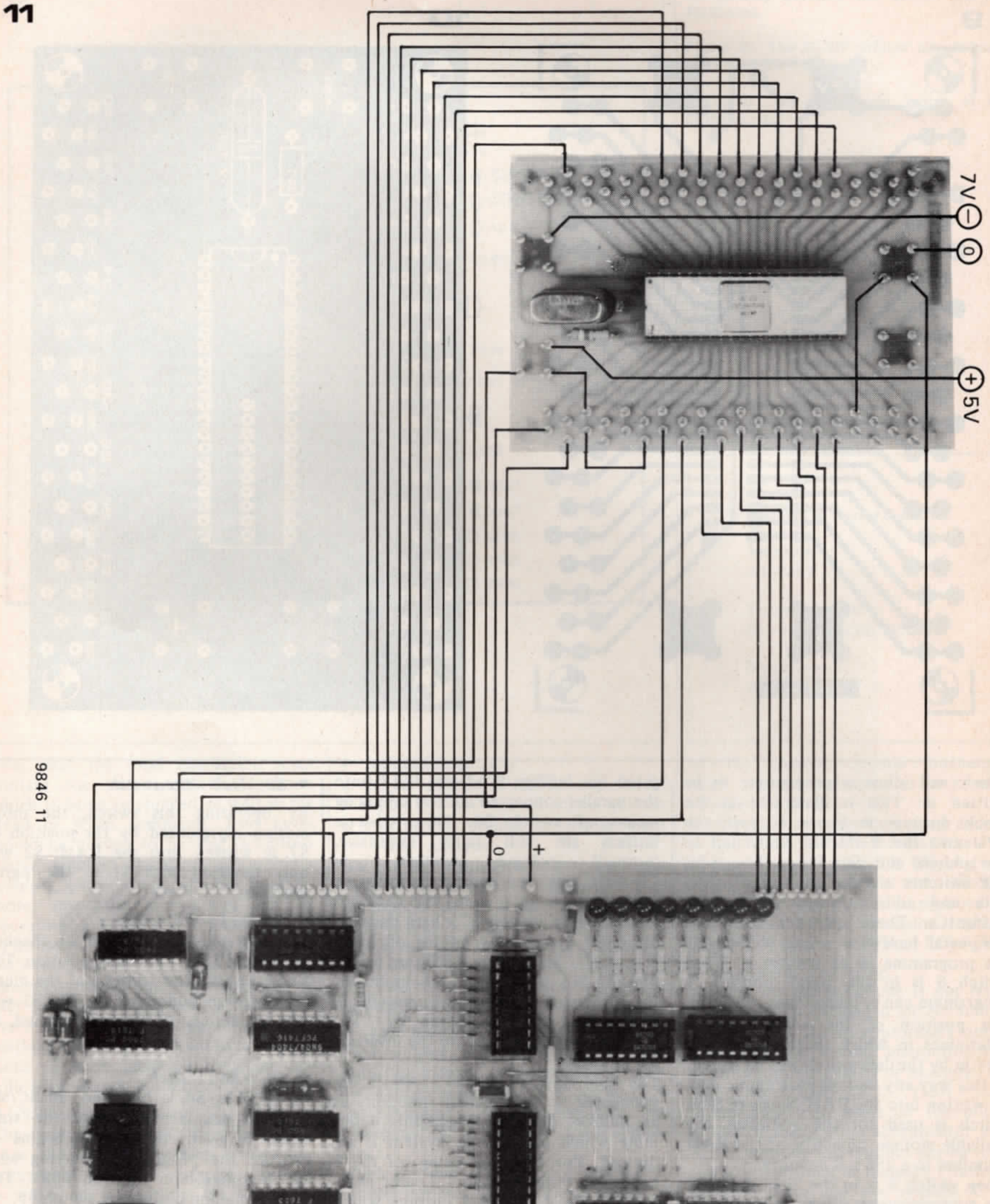
As soon as S4 is operated, all SC/MP registers are cleared. Once the start signal is given, the SC/MP begins to execute the programme, starting with the instruction in location ⌀⌀⌀1. The start command is given by switch S5.

- *S5, halt-reset switch*

Many programmes contain 'halt' instructions. In the case of such an instruction the SC/MP resets 'halt' flip-flop FF1, so that the CONT-input of the SC/MP is pulled low and the programme is halted. This condition is indicated by LED D9. Operating S5 sets FF1, so that the SC/MP recommences execution of the programme.

The circuit shown in figure 5 contains a large number of in- and outputs. Most of these must be connected direct to

11



12

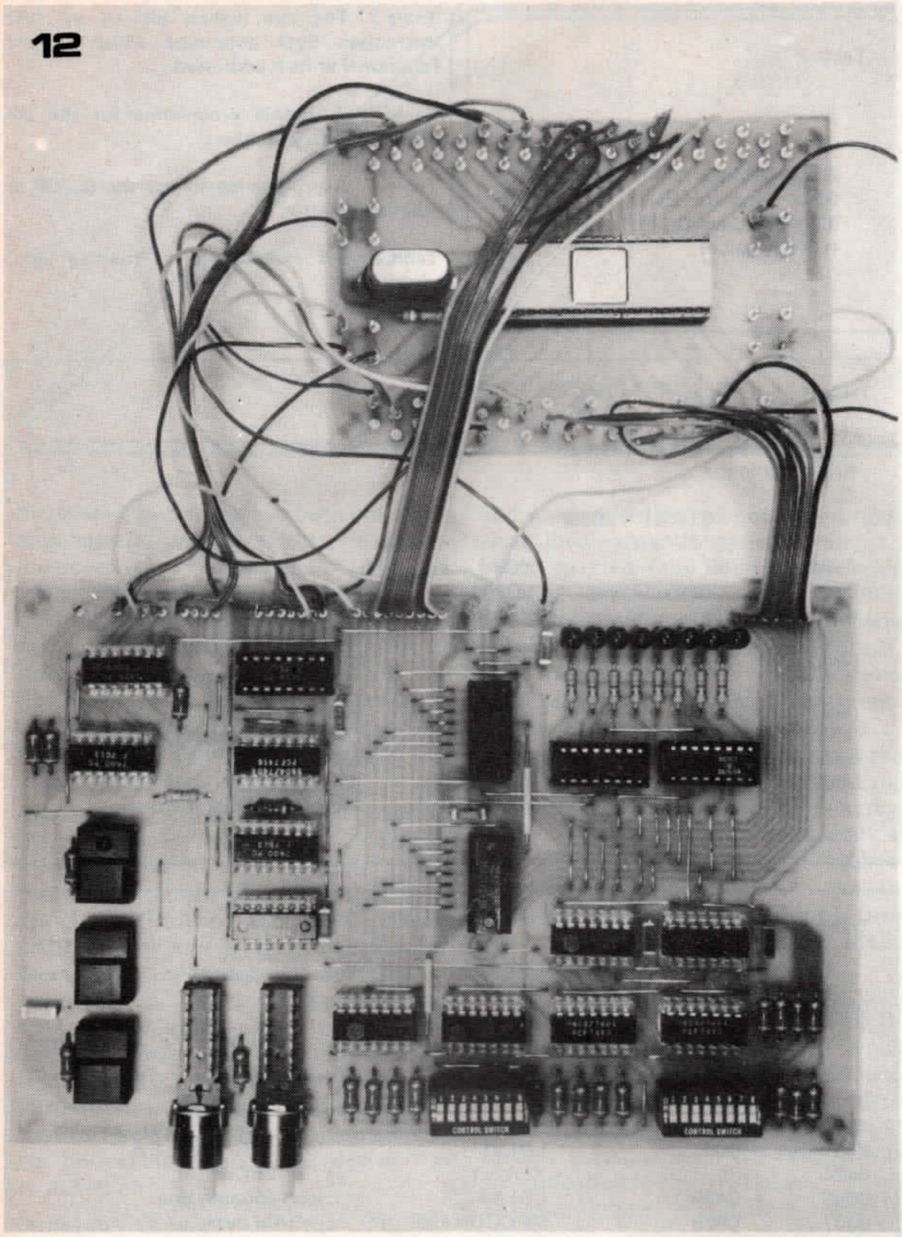


Figure 11. The wiring plan for the two circuit boards. All solder connections should be complete before mounting the MOS-ICs (that includes the SC/MP).

Figure 12. The complete 'hardwired' version of the system, ready to execute its first programme.

the SC/MP; the details of the various connections are shown in figure 8. Note that address-bit 1Ø of the RAM I/O-card is at logic '0'. Since this input is formed by the parallel connection of two TTL-inputs (pins 1 and 15 of IC1), it cannot be driven directly by the SC/MP. Fortunately, however, this does not present a problem, since only three peripheral addresses are required at present (see table 2).
For the time being, the card-enable input and output are simply interconnected. At a later stage the input can be used to disable the RAM-card, and this can prove extremely useful in the case of larger systems.

Construction

Two printed circuit boards were designed to accomodate the complete system. The circuit shown in figure 5, including the various switches, is mounted on the first board (see figures 6 and 7). The second board (see figures 9 and 10) takes the SC/MP, the quartz crystal and a resistor (R1). Since this board later becomes superfluous, it has a fairly 'general purpose' layout, thus enabling it to be reused for other applications. This does tend to complicate the wiring slightly, but if the wiring diagram in figure 11 is followed exactly there should be no problems.
The use of plug-in connectors is recommended, as this means that the 'hardware' can also be easily altered. How the completed version looks can be seen from figure 12 (and this month's cover picture).

Supply

As is apparent from the circuit diagram, two supply voltages, +5 V and −7 V, are necessary. The +5 V supply must be capable of delivering a current of at least 0.5 A, and the −7 V supply a current of approx. 100 mA. Both supplies should of course be stabilised. In both cases an IC-stabiliser is the best solution.
In view of future extensions to the circuit it is advisable to use a +5 V supply that can deliver a current of 1.5 A.

The first programmes

A prerequisite for operating the SC/MP is the SC/MP data sheet (Pub.-Nr. 420305227-001 A). When compiling a programme the above-mentioned 'SC/MP Instruction Guide' is also virtually indispensible. The programme examples assume that the user already has both these publications.
Table 3 shows a simple add programme arranged in the conventional programme form. The first column contains the address of the first instruction-byte of each instruction. NOP (no operation) is a 1-byte instruction, and the following byte therefore belongs to the next instruction. The second instruction is a 2-byte instruction, so that the first byte is stored at ØØØ1, the second byte at ØØØ2.
The second column contains the instructions and the operand addresses in (hexadecimal) machine code. The first two columns represent the results of the assembler programme. As mentioned earlier, an assembler programme can be used to translate a programme written in assembly language (the mnemonic letters and symbols in column 3) into the necessary logic (ones and zeros) that make up the machine code.
Column three contains the instructions and address symbols written in assembly language. Address symbols consist of a maximum of 6 randomly chosen letters or digits (without punctuation signs; the first symbol is always a letter). The assembler 'reads' the address symbols and calculates the required displacement values.
The last column contains an explanation of the programme steps; it has no effect upon the assembler.
As is apparent from table 2, when addressing the RAM, the LEDs or the data switch (DS), only the two highest address bits are decoded. This naturally

has certain consequences for the programming. For example, the LEDs will be referenced by every address beginning with Ø1. In the programme, PTR1 is used to address the LEDs, and PTR2 for the DS (only the 'higher' byte of the registers is used). In order to load the pointers, the information is first loaded into the AC, and the content of the AC is exchanged with that of the 'higher' pointer byte. The instruction ADDØ(2) causes the information represented by the position of the data switch to be added to the content of the AC (= ØØ); the result of this operation is stored in the AC. The next instruction results in the content of the AC being displayed by the LEDs. There then follows a HALT-instruction and the programme is interrupted. Another number may now be fed in via the data switch, and the programme restarted by means of the halt-reset switch. The next instruction is a jump command to 'LOOP'. LOOP is an address symbol, in this case a 'label', and labels are always followed by a colon (i.e. LOOP:).
The content of the PC is currently ØØØE, however this must be altered to ØØØ7 (back to LOOP). The content of the PC must therefore be reduced by ØØØ7 the value of the displacement of the jump-instruction thus becomes –Ø7 or F9 (two's complement).
A part of the programme is now executed for the second time, the 'new' DS information is added to the previous result and the outcome displayed by the LEDs.
Finally, tables 4 and 5 give two further programme examples. Table 4 contains a programme which converts the microprocessor to a (software) binary counter. The LEDs display the counter output, the count being continuously incremented by 1. The programme in table 5 is for a 'running light'; beginning from the left the LEDs light up in succession, the whole cycle being continuously repeated.
Although it may seem rather an over-investment in hardware, using a microprocessor to light up a few LEDs, the point is, of course, that these programmes are intended to illustrate the operating principles and programming techniques of the microprocessor.
Experimenting with programmes such as those given above is by far the best way to come to grips with the challenge of microprocessor technology.

(To be continued)

Literature:

1. *SC/MP data-sheet, pub. no. 420305227-001 A*
2. *SC/MP instruction guide, pub. no. 4200110 A*
3. *SC/MP technical description, pub. no. 4200079 A*
4. *SC/MP microprocessor applications handbook, pub. no. 420305239-001 A*
5. *SC/MP programming and assembler manual, pub. no. 4200094 B*

Table 2

```
ØØxx = address RAM
Ø1xx = address LEDs
Ø2xx = address DS (data switch)
(x = any value)
```

Table 2. The two highest bits of the first instruction byte determine which of the functional units is addressed.

Table 3. A simple programme for the addition of binary numbers.

Table 4. A programme to use the SC/MP as a binary counter.

Table 5. An example of a 'running light' programme.

Table 3

Add programme

```
                START = ØØØØ

ØØØØ     Ø8          NOP              ;
ØØØ1     C4Ø1        LDI Ø1           ; load Ø1 into AC
ØØØ3     35          XPAH 1           ; Ø1 in PTR 1 bit 8 . . . 15
ØØØ4     C4Ø2        LDI Ø2           ; load Ø2 into AC
ØØØ6     36          XPAH 2           ; Ø2 in PTR 2
                     LOOP:
ØØØ7     F2ØØ        ADD Ø (2)        ; add (EA) to (AC)
ØØØ9     C9ØØ        ST Ø (1)         ; (AC) in EA indicated by PTR 1,
ØØØB     ØØ          HALT             ; and stop
ØØØC     9ØF9        JMP LOOP         ; jump to LOOP
                     • END            ; end of assembler instructions
```

Table 4

Binary counter

```
                START = ØØØØ

ØØØØ     Ø8          NOP              ;
ØØØ1     ØØ          HALT             ;
ØØØ2     C4Ø1        LDI Ø1           ; load PTR 1 with EA
ØØØ4     35          XPAH 1           ;   of LEDs
ØØØ5     C4ØØ        LDI ØØ           ; load counter, one
ØØØ7     C8Ø9        ST COUNTER       ;   RAM byte
                     LOOP:
ØØØ9     A8Ø7        ILD COUNTER      ; increment counter
ØØØB     C9ØØ        ST Ø (1)         ; 'store' counter state in LEDs
ØØØD     8FFF        DLY X'FF         ; delay instruction
ØØØF     9ØF8        JMP LOOP         ; jump to LOOP
                     COUNTER:         ; label and instruction to
                     • BYTE               assembler to reserve
                                          a RAM byte
                     • END
```

Table 5

Running light

```
                START = ØØØØ

ØØØØ     Ø8          NOP              ;
ØØØ1     ØØ          HALT             ;
ØØØ2     C4Ø1        LDI Ø1           ; load EA of LEDs
ØØØ4     35          XPAH 1           ;   in PTR 1
ØØØ5     C4Ø1        LDI Ø1           ; load Ø1 into AC
                     LOOP:
ØØØ7     C9ØØ        ST Ø (1)         ; (AC) in LEDs
ØØØ9     1 E         RR               ; (AC) one bit to the right
ØØØA     Ø1          XAE              ; store (AC) in E
ØØØB     8FFF        DLY X'FF         ; delay
ØØØD     4Ø          LDE              ; (E) in AC
ØØØE     9ØF7        JMP LOOP         ; jump to LOOP
                     • END
```