experimenting with the SC/MP (2)

Among the topics discussed in this second part of the series on the SC/MP microprocessor are programming techniques, the SC/MP status register and address decoders. In addition a CPU card is described, which is designed to accomodate both the CPU itself and future 'monitor software'.

H. Huschitt

The majority of the SC/MP registers which are accessible to the programmer were discussed in the previous article (Elektor 31). However there remains an important 'multi-purpose' register still to be examined, and that is the status register.

Status register (SR)

The status register is an 8-bit register which, like the extension register, functions in conjunction with the accumulator (AC). By means of the instruction CSA (copy status to AC), the contents of the status register can be transferred to the AC, whilst the CAS-instruction (copy AC to status) does just the reverse.
A second similarity between the status register and the extension register is that in both cases a number of bits are available on external pins.
The functions of the various bits of the status register are shown in figure 1. From right to left: bits Ø, 1 and 2 are the so-called 'users flags' (FØ, F1 and F2). By means of instructions these flags can be set or reset. For example, the instruction LDI X'Ø2 followed by CAS results in flag 1 being set (i.e. storing '1'). This '1' is maintained until the contents of the AC, with a Ø in bit 1, are once more copied into the status register. Among other things, the three flags can be used in conjunction with a driver stage to directly control various peripherals such as lamps, relays, etc..
Bits 4 and 5 of the status register are the 'sense' inputs, i.e. Sense A (SA) and Sense B (SB) respectively. By means of these two inputs information up to two bits long can be transferred to the CPU using the CSA instruction. The contents of bits 4 and 5 are determined exclusively by the logic level of pins 17 and 18 of the SC/MP. The CAS instruction therefore has no effect upon these two bits.
An important function of the sense inputs is to set up programme loops. How this may be done is shown in table 1. First of all the contents of the status register are transferred to the AC. Then, one bit at a time, the contents of the AC and the byte ØØ1ØØØØØ are ANDed together, the result being once more stored in the AC. In the case where SB is '1', the result of the above operation will be ØØ1ØØØØØ.

```
Thus: (SR)              = xx1xxxxx
      (AC) after CSA    = xx1xxxxx
      AND with          = ØØ1ØØØØØ
      ----------------------------
      (AC)              = ØØ1ØØØØØ
```

(x = don't care)
If SB is '0' however, then the contents of the AC are also zero and a jump is performed to LABEL 1. The byte ØØ1ØØØØØ or X'2Ø is a 'mask', its function being to sift out the non-relevant bits. The above type of operation is a typical example of 'bit-handling', i.e. the manipulation of single bits.
Further examples of bit handling are:

- setting a particular bit of a byte and leaving the rest unaltered:

```
CSA         xxxxxxxx
ORI X'Ø4    ØØØØØ1ØØ
--------------------
CAS         xxxxx1xx
```

(x = don't care, but in this case also unchanged!)

- erasing one bit and leaving the rest unaltered:

```
CSA         xxxxxxxx
ANI X'FB    11111Ø11
--------------------
CAS         xxxxxØxx
```

- inverting a bit and leaving the rest unaltered:

```
CSA         xxxxxxxx
XRI X'Ø8    ØØØØ1ØØØ
--------------------
CAS         xxxx x̄xxx
```

In addition to being a sense input, SA can also function as an interrupt input. This is the case when bit 3 of the SR, the interrupt enable flag, is '1'. This flag can be set and reset by the instructions IEN (enable interrupt) and DINT (disable interrupt). The interrupt facility will be examined in greater detail at a later stage.
Bits 6 and 7 of the status register have an arithmetical function. Bit 7, the carry/link bit, is automatically set during all arithmetical manipulations as soon as there is a 1-carry from bit 7 of the AC. For example:

```
 1ØØØØØØØ   or   X'8Ø   or   -128
 1ØØØØØØØ        X'8Ø        -128
---------------------------------- +
1ØØØØØØØØ        X'1ØØ       -256
```

In this case the carry/link bit indicates whether the result is negative or not. If there is no carry from bit 7, then the carry/link bit is reset. The CY/L bit can

1

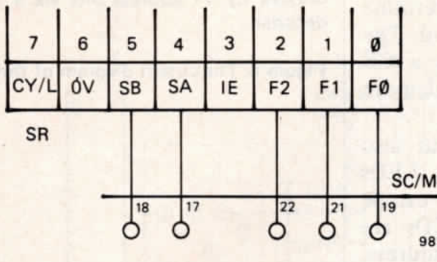
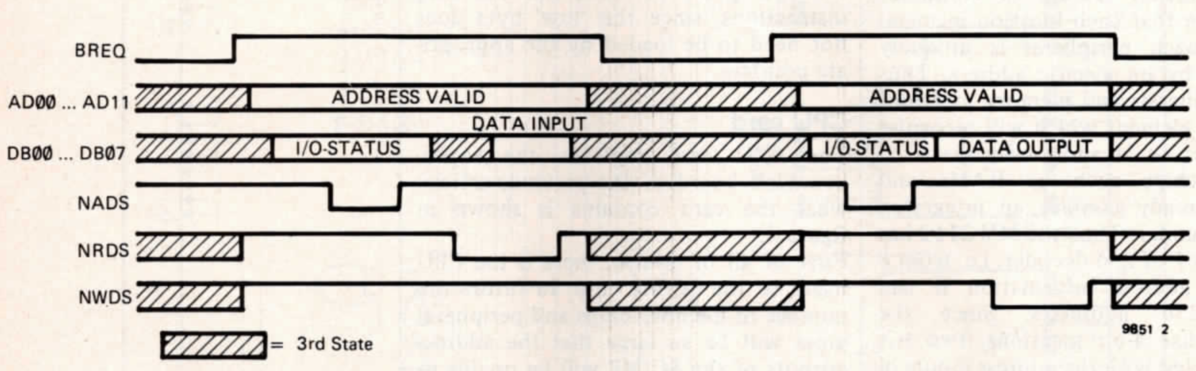


Table 1.

```
CSA          ; transfer (SR) to AC
ANI X'2Ø     ; AND the contents of AC
               and ØØ1ØØØØØ
JZ LABEL 1 ; If (SB) = Ø jump to
               label 1
LD.          ; If (SB) = 1 continue
               programme
```

2



be regarded as an extension to the left of the AC. In all arithmetical instructions the 'content' of the CY/L bit is added to the contents of the AC. The CY/L bit is set/reset by means of the instructions SCL (set carry/link) and CCL (clear carry/link). In the case of the instructions RRL (rotate right with link) and SRL (shift right with link), the CY/L functions as bit 8 of the AC.
Finally, bit 6 of the status register is the overflow bit (OV) which is automatically set as soon as there is a carry from bit 6 to bit 7 of the AC, and is reset in the absence of this carry. The overflow bit can be used to prevent calculational errors in the CPU. A good example is the addition of two positive numbers where the result might otherwise appear negative:

```
Ø1ØØØØØØ   or   X'4Ø   or    64
Ø1ØØØØØØ        X'4Ø         64   +
1ØØØØØØØ        X'8Ø       -128
```

I/O status on the data bus

In order to address a 64 k memory at least 16 address bits are necessary. However, as many readers have probably realised, the SC/MP has only 12 address lines. The remaining four bits are in fact multiplexed on the data bus.
During the NADS (negative address strobe), high-order address and status information, not data, is present on the 8-bit data bus. Table 2 shows how the 4 most significant address bits along with 4 status bits are multiplexed on the data bus, and figure 2 shows how the SC/MP controls the timing of the data input and output. The address information is only present on the data bus for a short period. To be able to address a memory (which is larger than 4 k) in this way it is obvious that this address information must be stored in an external register. The remaining four data bits which are available during the NADS are utilised as flags.
The R-flag is '1' when a read cycle is executed, and '0' for a write cycle. When the I-flag is '1' it signifies that the first byte of an instruction is being fetched. When the D-flag is '1' it indicates the fetch of the second byte of a delay instruction. Finally a '1' on the H-flag denotes the execution of a halt instruction.

Page structure of the memory

Neither the programme counter nor the pointers have an automatic carry from bit 11 (the 12th bit) to bit 12. This means that when the counter reaches X'ØFFF it increments not to X'1ØØØ, but to X'ØØØØ. In practice the next instruction is therefore fetched from address X'ØØØØ. The same holds true for the pointers. If e.g., the address X'5FFØ is stored in PTR1, then the instruction ST 1F (1) will not store the (AC) at address X'6ØØF (even though X'5FFØ + X'1F = X'6ØØF), but instead at address X'5ØØF. This also applies to auto-indexed addressing.
The four most significant bits of the PTRs and PC can only be altered by the instructions XPAH and XPPC. The

Table 2.

DB Ø	–	AD 12
DB 1	–	AD 13
DB 2	–	AD 14
DB 3	–	AD 15
DB 4	–	R-Flag
DB 5	–	I-Flag
DB 6	–	D-Flag
DB 7	–	H-Flag

Figure 1. Diagram showing the function of each bit of the status register and their pin connections.

Figure 2. SC/MP data in- and output timing.

Table 1. Specimen programme for testing a sense-bit.

Table 2. During the NADS the 4 most significant address bits along with 4 status bits are loaded onto the data bus in the following manner.

contents of these four bits form the 'page-address' of the memory. A section of memory, the addresses of which stretch from xØØØ up to and including xFFF, is called a page. The SC/MP is unable to 'turn' these pages by itself. Once it has read a page it will simply begin to read the same page again. An advantage of this type of page division is that a faulty programme occurring on one page cannot affect the information stored on the next. By means of the instructions WPAH and XPPC the programmer can reference the entire memory, which consists of 16 pages in all.

Address decoding and memory structure

A characteristic feature of computer structure is that each location in memory and each peripheral is uniquely identified by its specific address. Thus both peripherals and memory locations require an element which will recognise this address, i.e. an address decoder.
Memory chips such as RAMs and PROMs already contain an integrated address decoder. Thus the MM 2112 has an 8-bit to 1 of 256 decoder, i.e. from 8 bits of address information it can decode 256 addresses. Since the MM 2112 has 4-bit locations, two ICs are connected with the address inputs in parallel, in order to be able to process 8-bit data. The result is a 256 x 8 RAM.
Memory ICs are equipped with a $\overline{CE}$ (chip enable) or a $\overline{CS}$ (chip select) input. When this input is '1', the chip will not access data, despite address information being applied to the address inputs. Only when the input goes 'Ø' will the IC be enabled (since a 'Ø' enables the chip, the input is labelled $\overline{CE}$ or $\overline{CS}$, not CE or CS!). The $\overline{CE}$ or $\overline{CS}$ input can be controlled via an address decoder by the higher address bits. Figure 3 shows how address bits 8, 9 and 10 determine which of the 8 RAMs is addressed. The address decoder also possesses a $\overline{CE}$ input, thus permitting the use of more than one decoder.

Of course, not only RAMs, but also peripheral units such as the LEDs described in the previous article, can be addressed. If, however, the LEDs are required to respond to a single address, the help of an 8 by 1 to 256 decoder is needed.
If the LEDs are addressed solely by the higher address bits (via the address decoder) then this is known as 'incomplete address decoding'. From a software point of view, this can save instructions, since the 'low' byte does not need to be loaded by the appropriate pointers.

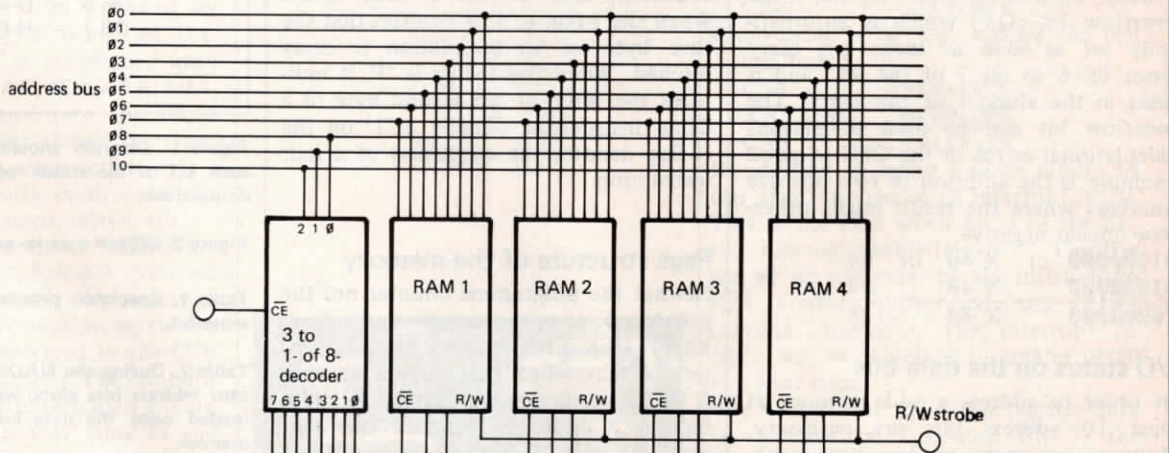
CPU card

The CPU card replaces the experimenter's board of the previous article; what the card contains is shown in figure 4.
First of all of course, there is the CPU itself, ie. the SC/MP chip. In future the number of memory chips and peripheral units will be so large that the address outputs of the SC/MP will be unable to drive them all. For this reason the address outputs are equipped with tri-state buffers. The analogue switches which were used in part 1 are here replaced by 'proper' tri-state buffers. This is because these switches have a certain transfer resistance, considerably limiting the fan-out. Thus they are unsuitable in a situation where a large amount of memory and peripheral units

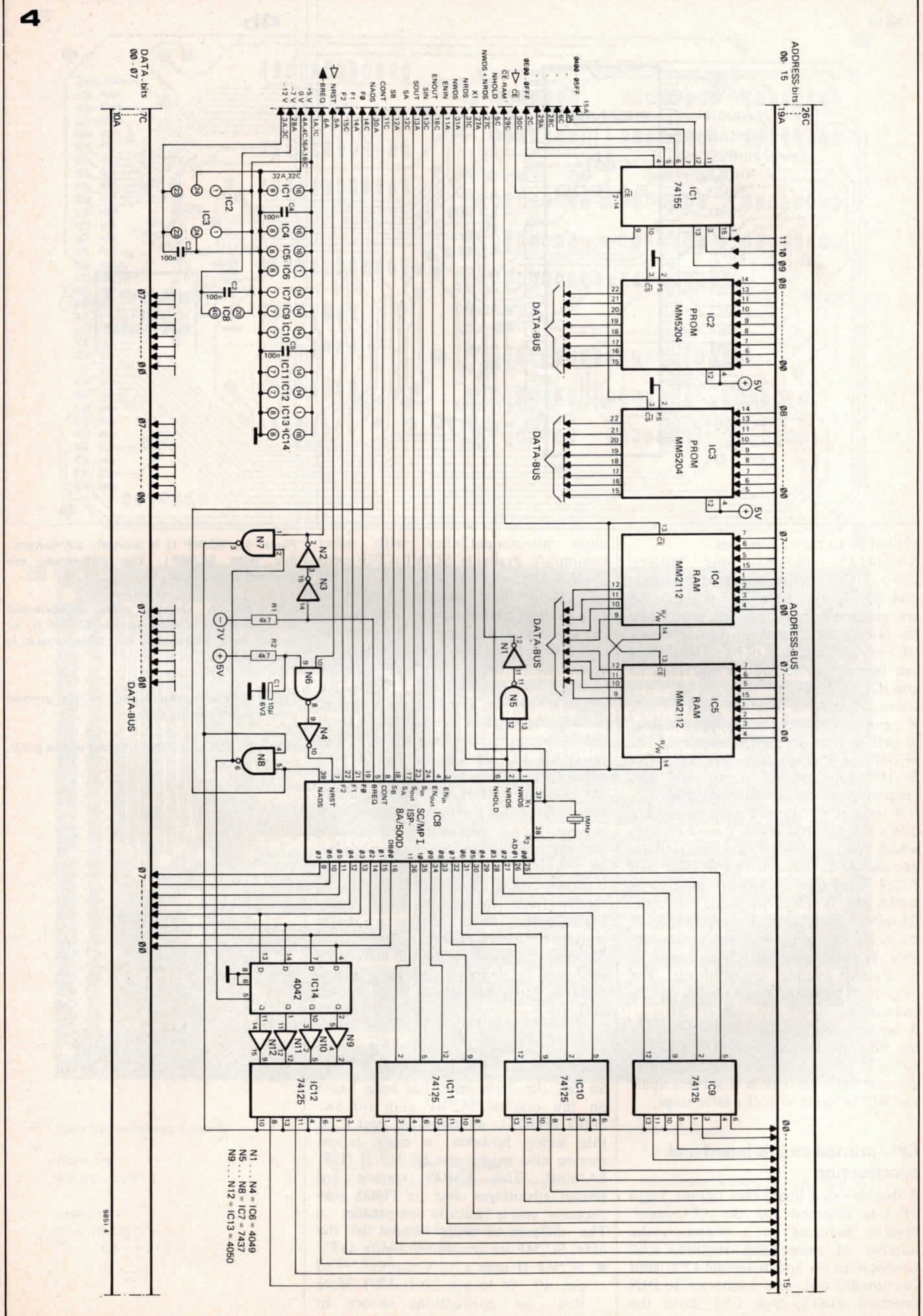
Figure 3. This figure provides an example of the use of address decoders. In this case 8 RAMs, each 256 x 8 bits, can be fully addressed by 11 address bits via a 3 to 1-of-8 decoder.

Figure 4. The circuit diagram of the CPU card.

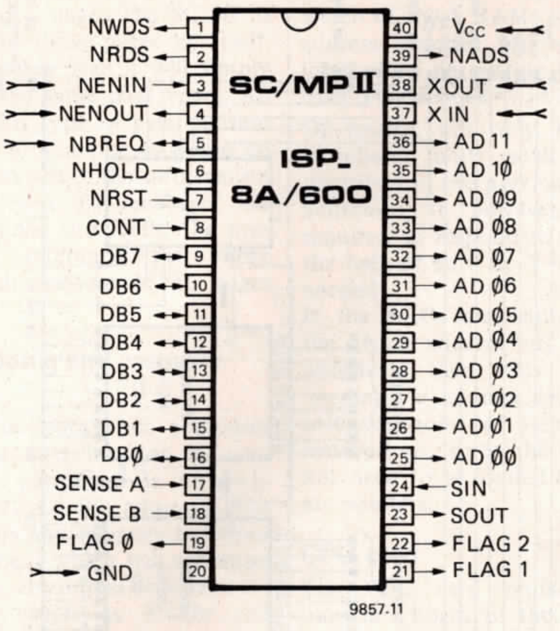
3



4


ADDRESS-bits Ø0 · 15
DATA-bits Ø0 · Ø7
IC1 74155
IC2 PROM MM5204
IC3 PROM MM5204
IC4 RAM MM2112
IC5 RAM MM2112
IC8 SC/MP I ISP-8A/500D
IC9 74125
IC10 74125
IC11 74125
IC12 74125
IC14 4042
DATA-BUS
ADDRESS-BUS
1MHz
4k7
10µ 6V3
100n
−7V
+5V
5V
N1 ... N4 = IC6 = 4049
N5 ... N8 = IC7 = 7437
N9 ... N12 = IC13 = 4050
9851 4

4a



4b

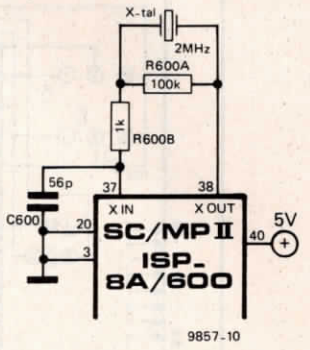


Figure 4a. SC/MP II is basically pin-compatible with SC/MP I. The differences are arrowed in this figure.

Figure 4b. A slightly more complicated timing circuit is required for SC/MP II, as shown here. Note that a 2 MHz crystal is now required!

Figure 5. The copper layout of the printed circuit board for the CPU.

Figure 6. The component layout of the p.c.b. for the CPU.

are linked to the address bus.
A 74125 contains four buffers. IC9 . . . IC11 are used to buffer address bits ØØ . . . 11. Address bits 12 . . .15 are multiplexed on the data bus during the NADS and stored on four flip-flops (IC14). The outputs of these flip-flops can be put on the address bus via IC13 and IC12.
Gates N4 and N6 together with C1 and R2 ensure that when the supply voltage is switched on the NRST input of the SC/MP is momentarily enabled. The SC/MP then begins to run the programme by jumping to address ØØØ1.
IC4 and IC5 together form a second 256 x 8 RAM. IC2 and IC3 are EPROMs which are addressed via an address decoder IC1. These two PROMs, each 512 x 8, together form a 1 k memory in which the monitor software is stored. 'Monitor' is generally understood to mean a programme designed to consideably facilitate various chores such as programme loading, debugging etc.. The subject of monitor programmes will be examined in detail in a later article.
A separate supply voltage is needed for the PROMs, making a total of three in all: –7, –12 and +5 V. Entering a programme into PROMs is a subject apart and will be dealt with at a later stage.

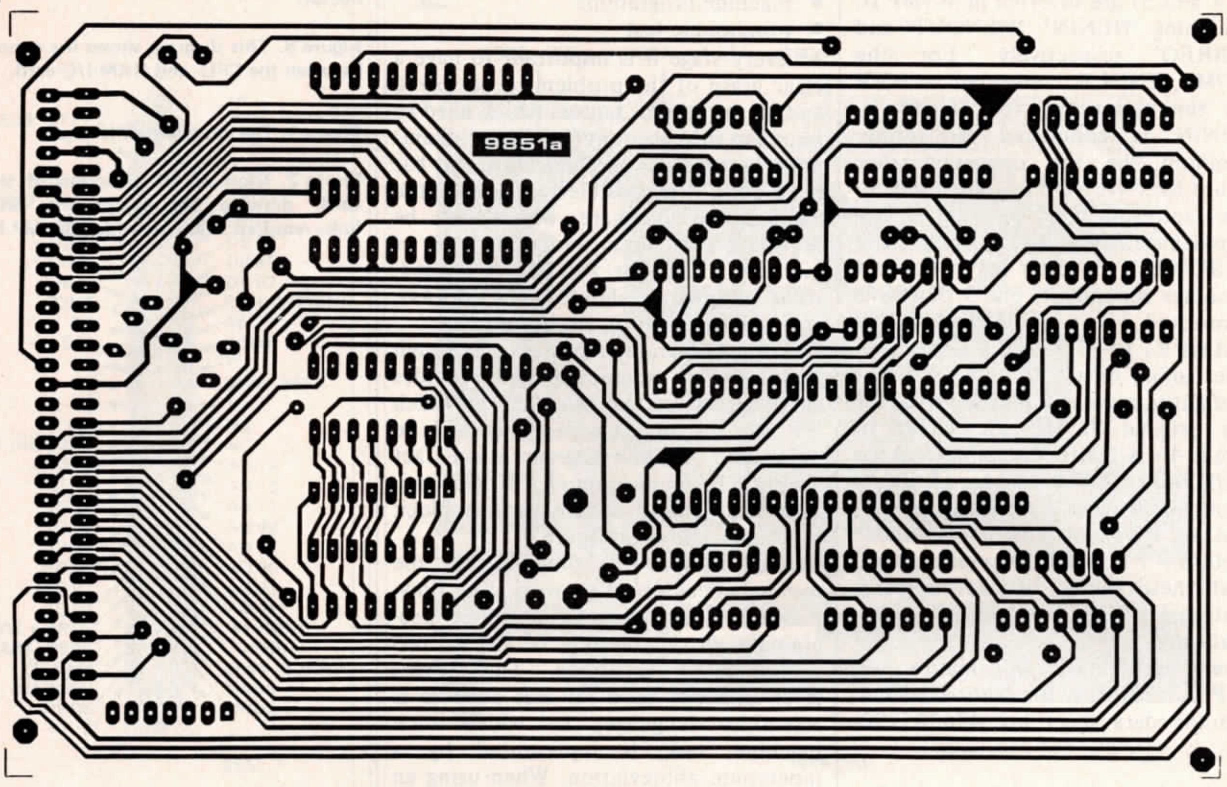
CPU printed circuit board and construction

A double-sided board (see figures 5 and 6) was designed for the CPU card, thereby reducing to a minimum the number of wire links required. All connections to and from the CPU card are brought out via a connector to DIN standard 41612, type C64. Both the wiring arrangement and the connector type were chosen with a view to optimum pin-compatibility with other commonly available SC/MP CPU cards (National ISP-8C/100 (E)).
The board has sufficient space to accomodate all the components shown in the circuit of figure 4. However at this stage it is not yet necessary to mount all of these components on the board. The RAMs and PROMs for example can be temporarily omitted, as can the address decoder IC1. Until this IC becomes necessary the 74155 of the RAM I/O card, which has become redundant, may be used instead. As yet, the tri-state buffers are not absolutely necessary either; however since they require through connections to be made on the board they are best mounted immediately.
It is important to use good quality IC sockets and to ensure good solder connections, since tracing a faulty contact is no easy matter.
The pin configuration of the connector is shown in figure 7, and the connections to the RAM I/O card are shown in figure 8.

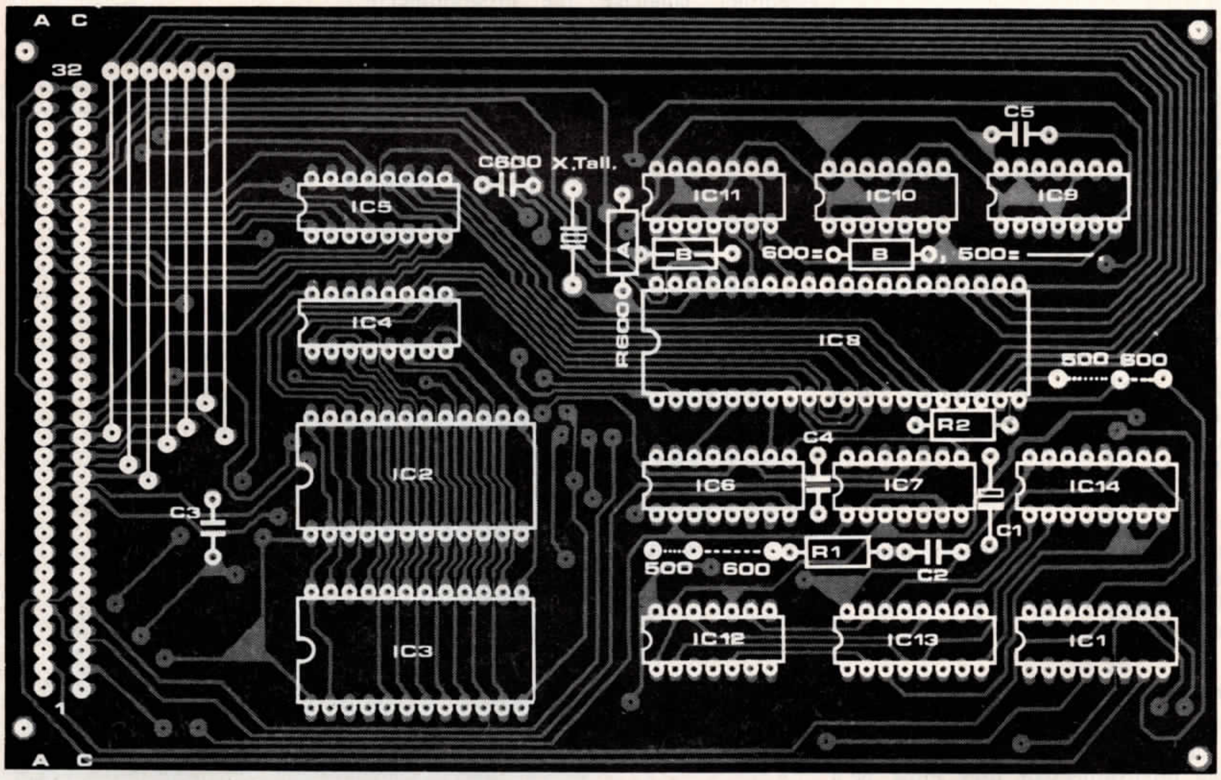
SC/MP II and the p.c.board

So far, the description has been based on the original SC/MP chip ISP-8A/500D (PMOS). As noted in part 1 of this series, however, a more recent version also exists: the SC/MP II (ISP-8A/600). This NMOS version has several advantages over its PMOS predecessor, and is basically compatible.
The differences with respect to the older SC/MP are arrowed in figure 4a:
- SC/MP II uses a +5 V supply: '+' to pin 40, '0' to pin 20 (GND). Note that the connections shown in figure 4 apply to the original version of the SC/MP, not to SC/MP II!

5



6



Parts list to figures 4 and 6

Resistors:
R1,R2 = 4k7

Capacitors:
C1 = 10 μ/6.3 V
C2 . . . C5 = 100 n

Semiconductors:
IC1 = 74155
IC2,IC3 = MM 5204 (National)
IC4,IC5 = MM 2112 (National) or equiv.
IC6 = 4049
IC7 = 7437
IC8 = ISP-8 A/500 D (SC/MP)
IC9 . . . IC12 = 74125
IC13 = 4050
IC14 = 4042

Miscellaneous:
Xtal = 1 MHz crystal

Modifications when using SC/MP II:
R600A = 100 k
R600B = 1 k
C600 = 56 p
IC8 = ISP-8A/600 (SC/MP II)
Xtal = 2 MHz crystal

- 'ENIN', 'ENOUT' and 'BREQ' (pins 3, 4 and 5) are inverted in SC/MP II, becoming 'NENIN', 'NENOUT' and 'NBREQ' respectively. For the SC/MP, ENIN is connected to +5 V via the connector; for SC/MP II, NENIN is connected to supply common via the connector (see figure 8). (N)ENOUT is not used, so the fact that it is inverted is unimportant. Since BREQ is inverted in SC/MP II, inverter N3 in figure 4 becomes redundant; pin 5 of IC8 is connected direct to the input of N2 and via R1 to +5 V (not −7 V!).
- The timing circuit for the SC/MP II is slightly more complicated than for the original SC/MP. As shown in figure 4b, a 2 MHz crystal is used for SC/MP II; a 100 k resistor (R600A) is added in parallel with the crystal; and an RC-filter network is added: R600B = 1 k and C600 = 56 p. Note that these values differ from early National Semiconductor information.

Further preliminary information on SC/MP II is given in the National Semiconductor data sheet no. 426305290-001A.

The printed circuit board is suitable for both SC/MP and SC/MP II. Photo 1 shows the board with SC/MP II mounted; photo 2 is a close-up of the section where modifications are required if the older SC/MP is to be used.
For the SC/MP (ISP-8A/500D), C600 and R600A are omitted; R600B is replaced by a wire link; the two other wire links are mounted in the position labelled '500'; a 1 MHz crystal is used; ENIN (pin 11A of the connector) is connected to +5 V; pin 28A of the connector is connected to −7 V.
For SC/MP II (ISP-8A/600), C600, R600A and R600B are mounted; the wire links are mounted in position '600' (see photo 2); a 2 MHz crystal is used; NENIN is connected to supply common; pin 28A of the connector is connected to +5 V.
Provision is made on an extension board that is to be published next month for selecting either a +5 V or a −7 V supply for pin 28A of the connector.

Software

However great his knowledge of computer circuitry or hardware, the prospective microprocessor user must be able to 'speak' a computer language in order to be able to communicate with the machine. Since the task of actually writing a programme represents by far the most difficult problem with which the user will be faced, a good deal of space will be devoted to the question of programming skills.
Writing a programme is considerably simplified if the problem is approached in the following systematic fashion:

- problem definition
- hardware concept
- flow-diagram
- source programme
- machine programme
- programme test

At every stage it is important to have a clear grasp of the problem to be solved and all the various factors which need to be taken into account (i.e. the problem) must be precisely defined. Once this has been done it is possible to calculate the amount of hardware which will be required; how much and what type of memory (RAM or PROM) is needed, what peripherals should be used (AD, DA- converters etc.), and so on.
The flow-diagram represents a rough draft of the programme and provides an overview of the sequence in which the various programme steps will be executed. A flow-diagram is in fact simply a block diagram of a programme. The significance of the various block symbols is shown in figure 9.
Having compiled a flow-diagram the rest is basically routine work. The flow-diagram is replaced by a source programme written in one of the various programming languages. The lowest level language used for this purpose is assembler language, in which each machine task is represented by a mnemonic abbreviation. When using an assembler language the programmer must be familiar with the machine structure.

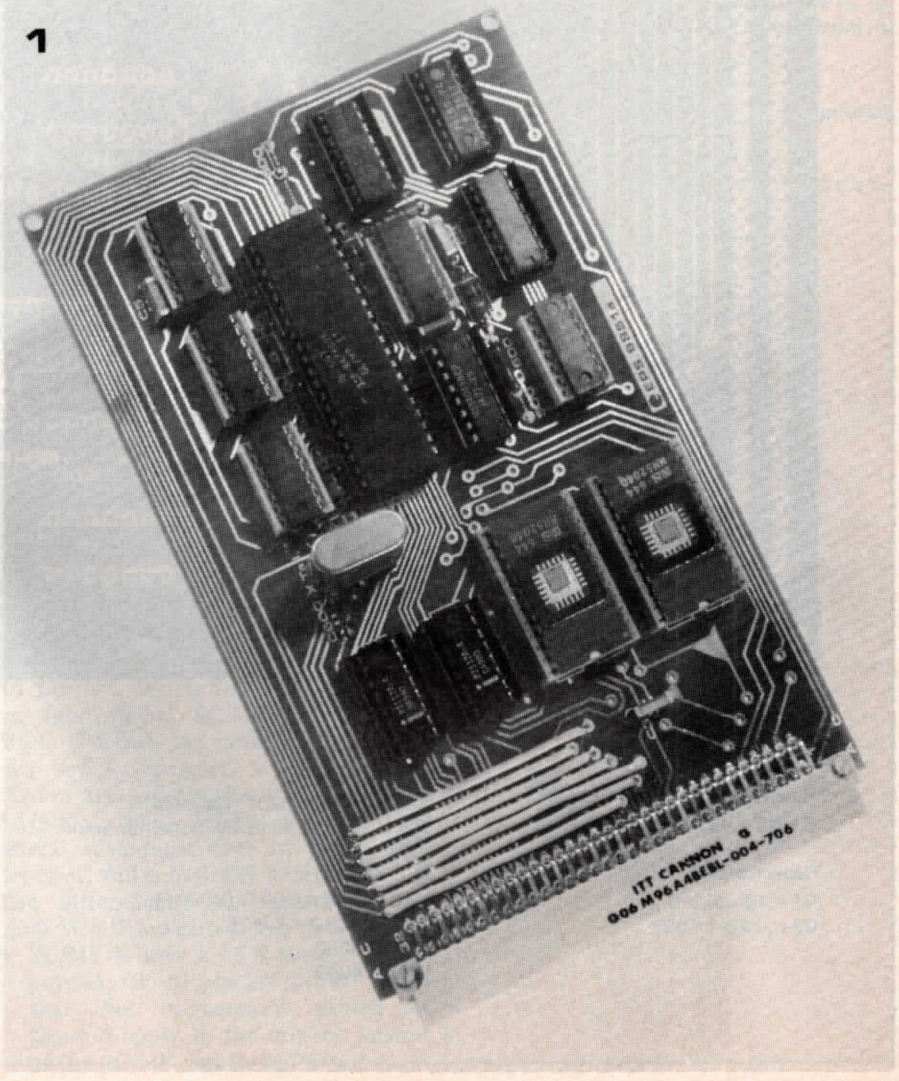
Figure 7. The pin configuration of the connector.

Figure 8. This diagram shows the connections between the CPU- and RAM-I/O-card.

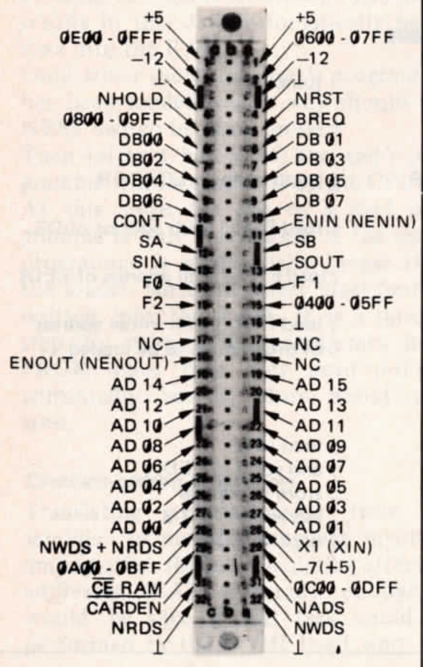
Photo 1. The complete CPU card.

Photo 2. Close-up of a section of the CPU card, showing the components and wire links required when mounting SC/MP II.

1



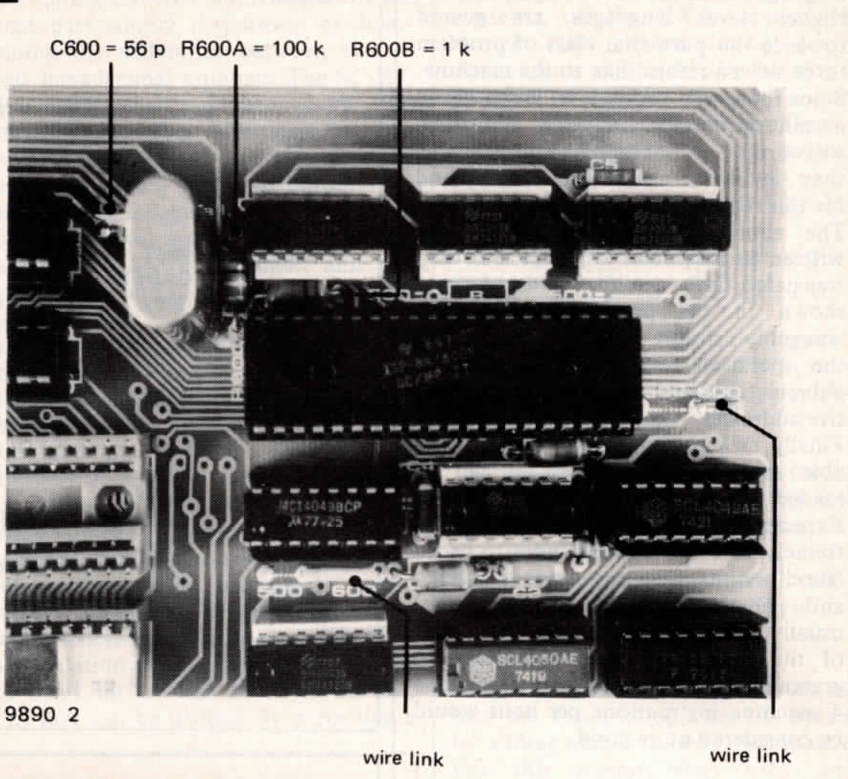
7



9851.7

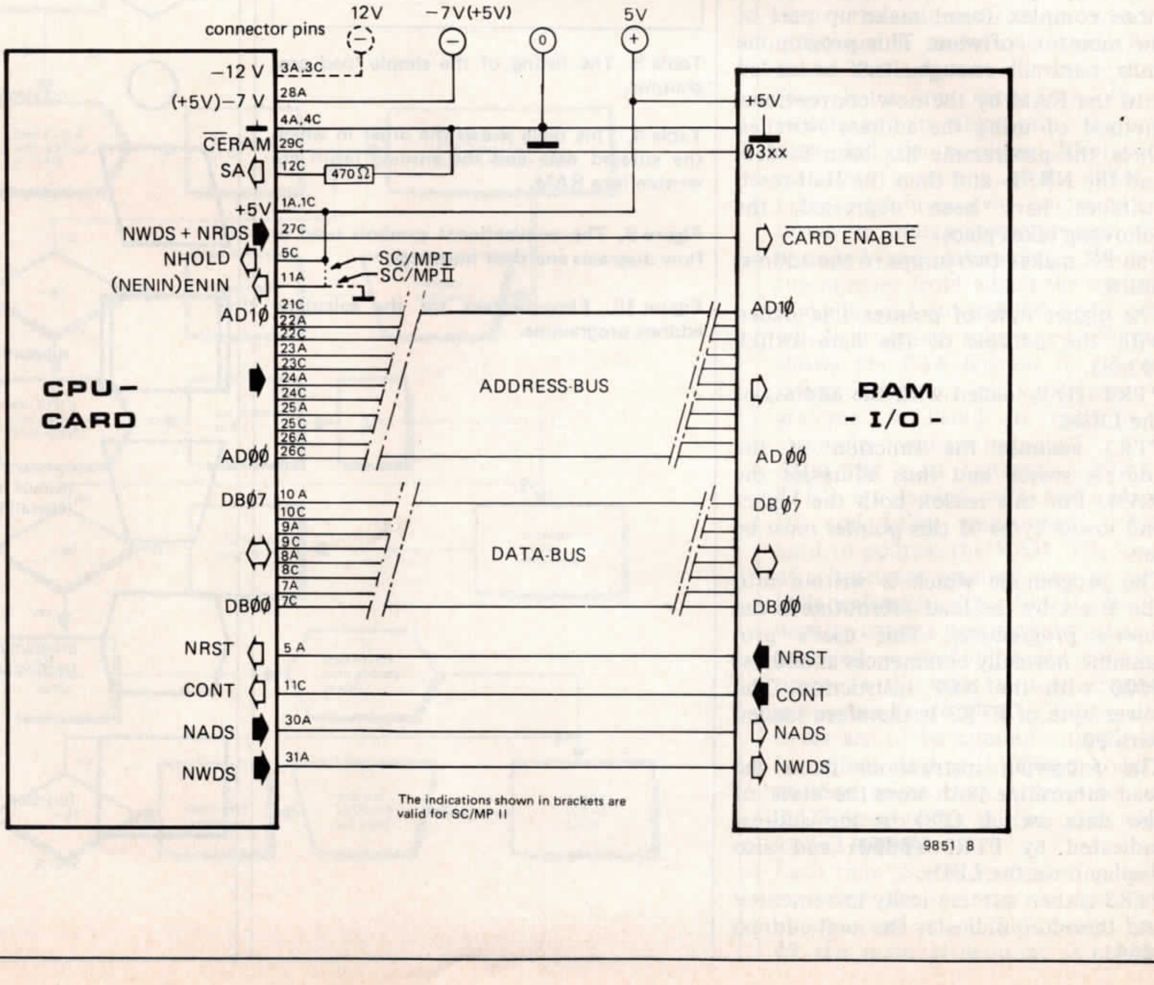
The indications shown in brackets are valid for SC/MP II

2



9890 2

8



The indications shown in brackets are valid for SC/MP II

9851 8

This is not the case with higher level languages such as, for example, BASIC. Higher level languages are geared towards the particular class of problem to be solved rather than to the machine. Since the use of higher level languages in a microprocessor requires additional software, i.e. a compiler programme, they involve considerable outlay, and for this reason are not discussed here.
The source programme is therefore written in assembler language and then translated into machine code. As was shown in the previous article, this 'assembly' simply consists of filling in the operation code for the mnemonic abbreviations and calculating the effective addresses.
Finally, when the programme is available in machine language, it can be loaded into memory and tested. Experience has shown that it is extremely rare for a programme to prove 'good' on its first run. Indeed tracing and eliminating faults (debugging) usually accounts for a considerable part of the time taken to develop a programme. An average of something like 4 machine instructions per hour would be considered quite good . . .

Helpful programmes

The operation of development systems can be considerably simplified by the use of a few special programmes.
At this stage a simple programme can assume the function of the address switch. At a later date, the programme for this purpose (see table 3) will, in a more complex form, make up part of the monitor software. This programme must, naturally enough, itself be loaded into the RAM by the now conventional method of using the address switches. Once the programme has been loaded, and the NRST- and then the Halt-reset-switches have been depressed, the following takes place:
The PC makes two jumps to the address ØØEB.
The higher byte of pointer 1 is loaded with the address of the data switch (Ø2xx).
PTR2 (H) is loaded with the address of the LEDs.
PTR3 assumes the function of the address switch and thus addresses the RAM. For this reason both the higher and lower bytes of this pointer must be set.
The programme which is written into the RAM by the load subroutine is the 'user's programme'. This user's programme normally commences at address ØØØØ with the NOP instruction. The lower byte of PTR3 is therefore loaded with ØØ.
The following instructions from the load subroutine both store the 'state' of the data switch (Ø8) in the address indicated by PTR3 (ØØØØ) and also display it on the LEDs.
PTR3 is then automatically incremented and therefore indicates the next address (ØØØ1).

Table 3.

```
                                START = ØØØØ
ØØØØ        Ø8             NOP
ØØØ1        9Ø7F           JMP 7F                ; jump
                           •
                           •
                           •
ØØ82        9Ø67           JMP LOADER            ; jump again to LOADER
            LOADER:
ØØEB        C4Ø2           LDI H (SB)            ; load PTR 1 with address of DS
ØØED        35             XPAH 1
ØØEE        C4Ø1           LDI H (LED)           ; load PTR 2 with address of LEDs
ØØFØ        36             XPAH 2
ØØF1        C4ØØ           LDI L (ADR)           ; load PTR 3 with initial address
ØØF3        33             XPAL 3                ; of programme to be loaded
ØØF4        C4ØØ           LDI H (ADR)
ØØF6        37             XPAH 3
            LOOP:
ØØF7        C1ØØ           LD Ø (1)              ; load (DS)
ØØF9        CAØØ           ST Ø (2)              ; and store in LEDs
ØØFB        CFØ1           ST@1 (3)              ; store in address
ØØFD        ØØ             HALT                  ; increment PTR 3
ØØFE        9ØF7           JMP LOOP              ; back to LOOP
                           * END
```

Table 4.

ØØ3F	Minuend, Lower Byte
	Minuend, Higher Byte
	Subtrahend, Lower Byte
	Subtrahend, Higher Byte
	Difference Lower Byte
	Difference Higher Byte

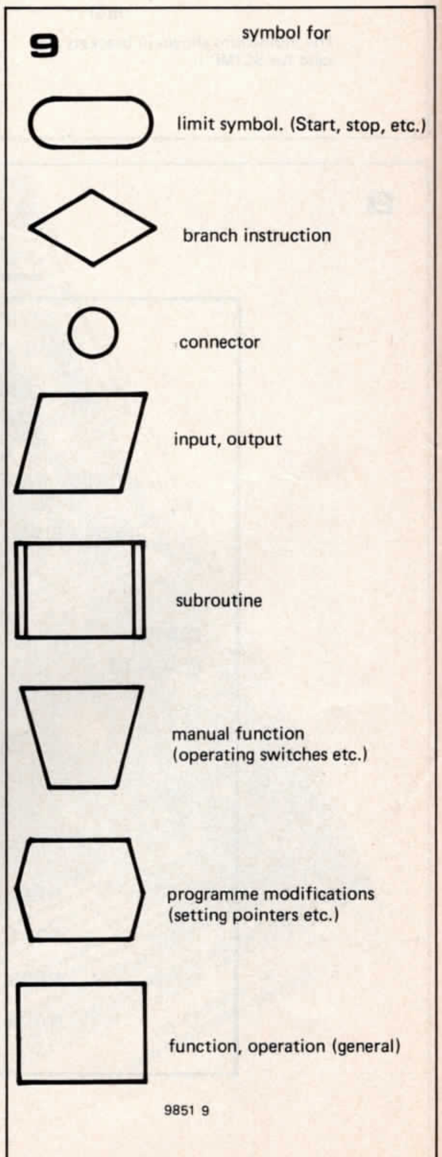
Table 3. The listing of the simple load programme.

Table 4. This table shows the order in which the entered data and the ensuing result are written into RAM.

Figure 9. The conventional symbols used in flow-diagrams and their meanings.

Figure 10. Flow-diagram for the calculate-address programme.

9


symbol for
limit symbol. (Start, stop, etc.)
branch instruction
connector
input, output
subroutine
manual function (operating switches etc.)
programme modifications (setting pointers etc.)
function, operation (general)
9851 9

The CPU is now stopped.
By means of the data switch the data for the next memory location of the user's programme can be set.
Pressing the halt-reset switch once more results in this data automatically being read into the RAM.
Only when the entire user's programme has been loaded in this way should the NRST switch be again pressed.
Then (after a halt-reset) the user's programme will be executed by the CPU.
At this stage the use of a load programme is only worthwhile if the user's programme is substantially longer than the loader, since the latter must first be written into the RAM. It is a natural step to store such programmes in a PROM since then they need not be continually reread. More about this later.

Calculate-address-programme

Translating a programme from assembler- to machine language involves, among other things, calculating effective addresses (EA). Naturally enough it would be nice if this task could be performed by the SC/MP itself, and, not surprisingly, this it can do.
Calculating effective addresses is for the most part simply a question of determining the difference between two 4-digit hexadecimal numbers. The SC/MP, like many other microprocessors, cannot calculate the difference between two numbers directly, but requires the assistance of an 'add' instruction. The instructions recognised by the SC/MP for this purpose are: CAD (complement and add), CAI (complement and add immediate) and CAE (complement and add extension).
These instructions will result only in the complement (all bits are inverted) of the addressed data being added to the contents of the AC, which of course does not give the difference of the two numbers. In order to obtain a difference the two's complement of the appropriate number is needed.
As is well known, the two's complement can be obtained by adding a '1' to the one's complement. It was stated earlier in the article that during all arithmetical manipulation of data the 'contents' of the CY/L bit are also added to the AC. This fact can be utilised by preceding a complement instruction with an SCL instruction (set carry link), so that the contents of the CY/L appear as a number ØØØ1, which will be added to the contents of the AC. In this way it is possible to obtain the two's complement of a number and therefore the difference between it and a second number.
For example: X'55 – X'Ø3 = X'52

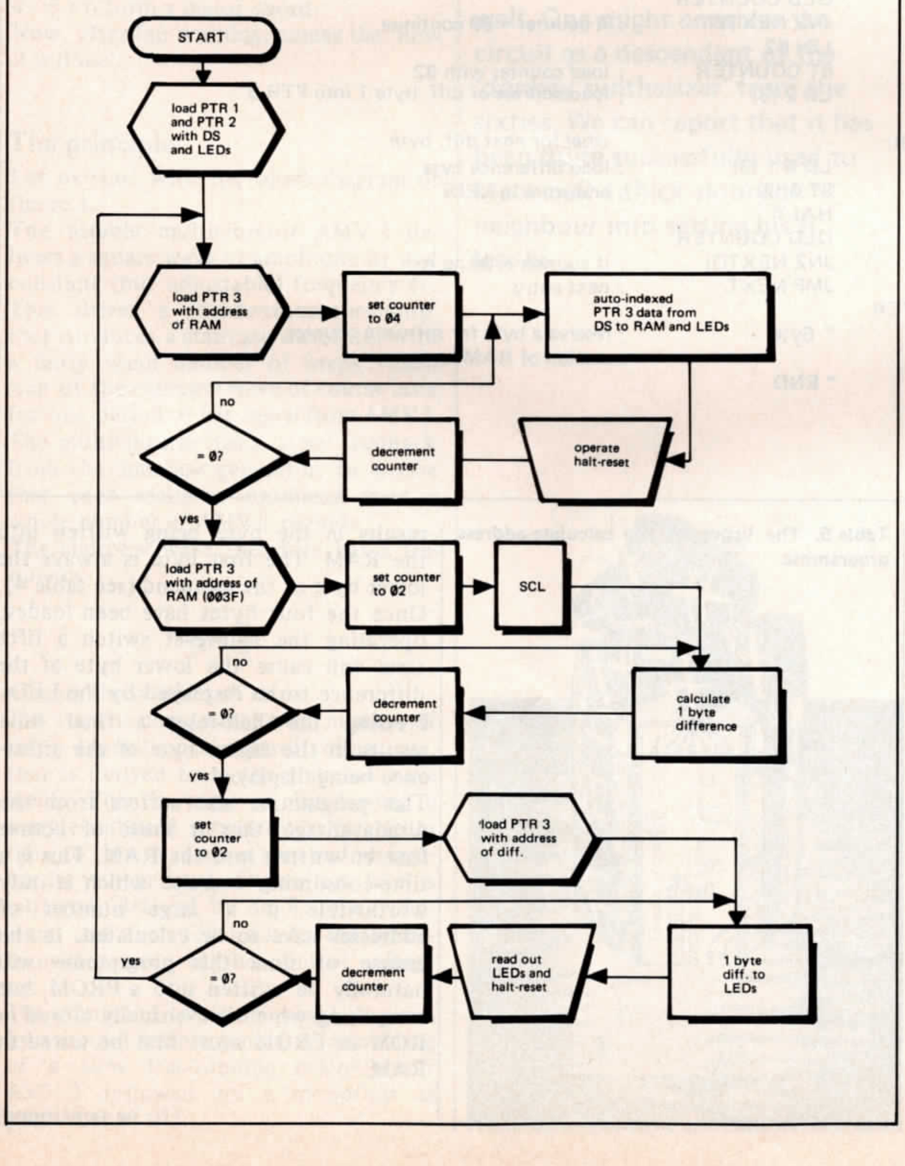
```
Ø1Ø1Ø1Ø1
111111ØØ (one's complement of Ø3)
ØØØØØØØ1 (contents of CY/L)
______________________________ +
Ø1Ø1ØØ1Ø
```

The 1-carry from the most significant bit is once more stored in CY/L, since it may be required in multi-bit calculations such as the following:
X'5555 – X'ØØØ3 = X'5552
The lower bytes are handled in the same way as the previous example. The higher bytes are manipulated as follows:

```
Ø1Ø1Ø1Ø1
11111111 (one's complement of ØØ)
ØØØØØØØ1 (carry from lower byte in
          CY/L)
______________________________ +
Ø1Ø1Ø1Ø1
```

In the case of both 8- and 16-bit numbers the two's complement is obtained by adding one to the one's complement. For this reason, even with 2-byte numbers only the first complement instruction is preceded by an SCL instruction. If CY/L remains set for the higher byte operation this means there has been a carry from the lower byte.
So much for how the SC/MP actually performs the arithmetical operations; however, it is of course necessary to enter the addresses from which the difference is to be calculated. Table 4 shows the sequence in which the first four bytes are read from the data switch, and in which order the entered data and the resulting difference are read into a section of the RAM. This table introduces two commonly used terms in micro-programming, namely minuend and subtrahend. Minuend is the number from which the subtraction is made, and subtrahend is the number which is being subtracted. Figure 10 shows the flow-diagram for the calculate-programme, and the same programme is listed in machine- and assembler-language in table 5.
After the start instruction PTR1 and PTR2 are loaded with the address of DS and LEDs respectively. PTR3 is used to address the RAM. The function of a 'software counter' may as yet be a little unfamiliar. This type of counter is used to cause a specific subroutine to be repeated a predetermined number of times. In this case the load programme is executed a total of 4 times, since 4 bytes are to be entered into the RAM. In actual fact a software counter is nothing more than a location in the RAM reserved for this purpose, which is loaded with Ø4.
Each time the load programme has been executed the counter is decremented by one, and when the counter reaches ØØ the main programme is continued.

10



START
load PTR 1 and PTR 2 with DS and LEDs
load PTR 3 with address of RAM
set counter to Ø4
auto-indexed PTR 3 data from DS to RAM and LEDs
operate halt-reset
decrement counter
= Ø?
no
yes
load PTR 3 with address of RAM (ØØ3F)
set counter to Ø2
SCL
calculate 1 byte difference
decrement counter
= Ø?
no
yes
set counter to Ø2
load PTR 3 with address of diff.
1 byte diff. to LEDs
read out LEDs and halt-reset
decrement counter
= Ø?
no
yes

Table 5.

```
                       START = ØØØØ
ØØØØ     Ø8            NOP
ØØØ1     C4Ø2          LDI Ø2              ; load PTR 1 with address of DS
ØØØ3     35            XPAH 1
ØØØ4     C4Ø1          LDI Ø1              ; load PTR 2 with address of LEDs
ØØØ6     36            XPAH 2
         NEXT:
ØØØ7     C43F          LDI L (RAM)
ØØØ9     33            XPAL 3              ; load PTR 3 with address of RAM
ØØØA     C4ØØ          LDI H (RAM)
ØØØC     37            XPAH 3
ØØØD     C4Ø4          LDI Ø4              ; load counter with Ø4
ØØØF     C82E          ST COUNTER
         LDDS:                             ; label for load programme
ØØ11     C1ØØ          LD Ø (1)            ; load DS
ØØ13     CAØØ          ST Ø (2)            ; and store in LEDs
ØØ15     CFØ1          ST@1 (3)            ; and store in RAM
ØØ17     ØØ            HALT
ØØ18     B825          DLD COUNTER         ; decrement counter
ØØ1A     9CF5          JNZ LD DS           ; if counter ≠ ØØ fetch following byte from DS
         SUBTR:
ØØ1C     C7FC          LD @ — 4 (3)        ; reload PTR 3 with address of RAM
ØØ1E     C4Ø2          LDI Ø2
ØØ2Ø     C81D          ST COUNTER          ; load counter with Ø2
ØØ22     Ø3            SCL                 ; set carry/link fot two's complement
         NEXTBY:
ØØ23     C7Ø1          LD @ 1 (3)          ; load minuend
ØØ25     FBØ1          CAD 1 (3)           ; complement subtrahend and add
ØØ27     CBØ3          ST 3 (3)              write difference in RAM
ØØ29     B814          DLD COUNTER
ØØ2B     9CF6          JNZ NEXTBY          ; if counter = ØØ continue
ØØ2D     C4Ø2          LDI Ø2
ØØ2F     C8ØE          ST COUNTER          ; load counter with Ø2
ØØ31     C7Ø2          LD 2 (3)            ; load address of diff. byte 1 into PTR 3
         NEXTDI:                             label for next diff. byte
ØØ33     C7Ø1          LD @ 1 (3)          ; load difference byte
ØØ35     CA ØØ         ST Ø (2)            ; and store in LEDs
ØØ37     ØØ            HALT
ØØ38     B8Ø5          DLD COUNTER
ØØ3A     9CF7          JNZ NEXTDI          ; if counter = ØØ go to
ØØ3C     9ØC9          JMP NEXT            ; next entry
         COUNTER
                       * Byte              ; reserve a byte for software counter
         RAM                               ; section of RAM for entry
                       * END
```

Table 5. The listing of the calculate-address programme.

PTR3 is once more loaded with the RAM address (ØØ3F), and the counter is now loaded with Ø2. The following section of the programme is the calculate-difference routine and this has to be executed twice, once for the lower byte and once for the higher byte. When that is completed the counter is once more loaded with Ø2, since the calculated differences must be displayed in turn by the LEDs; first the lower byte and then the higher byte. The 'display routine' must therefore be executed twice. When the counter reaches ØØ, fresh data can once more be written into the RAM.

Actually running the entire calculate-address-programme is a simple matter. Once the programme as shown in table 5 has been loaded into the RAM the NRST is operated. The first byte is then entered on the data switch. A single operation of the halt-reset switch results in the byte being written into the RAM. The first byte is always the lower byte of the minuend (see table 4). Once the four bytes have been loaded, operating the halt-reset switch a fifth time will cause the lower byte of the difference to be displayed by the LEDs. Pressing the halt-reset a final time results in the higher byte of the difference being displayed.

This programme also suffers from the disadavantage that it must of course first be written into the RAM. This is a time-consuming exercise which is only worthwhile if a large number of addresses have to be calculated. In the course of time this programme will naturally be written into a PROM, but everything which is eventually stored in ROM or PROM must first be tested in RAM.

(to be continued)