experimenting with the SC/MP (3)

This, the third article in the SC/MP series, introduces the memory extension card, which, in addition to containing ¾ k of RAM and ½ k of PROM, also houses the multiplexer and priority encoder. The latter hardware allows the SC/MP to handle interrupt requests from more than one peripheral device. The article also examines the software involved in interrupt operations.

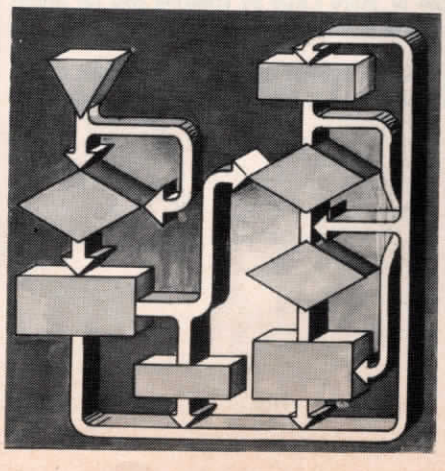
H. Huschitt

Interrupt operations

An interrupt operation occurs when an externally generated control signal causes the SC/MP temporarily to suspend main programme execution. The interrupt request will typically be issued by a peripheral device such as, e.g., a display which requires refreshing. When the CPU acknowledges such an interrupt, it jumps from the main programme to a special routine to service the interrupting device – after saving the return-to-main programme address. Once the interrupting device has been serviced, the CPU automatically resumes main programme execution. This process is illustrated in figure 1. Note that, in principle, an interrupt routine is quite similar to a subroutine call, except that the jump is initiated externally by an interrupt request rather than internally by an instruction in the current programme.

The situation becomes more complicated when the CPU receives several interrupt requests more or less simultaneously and has to choose between more than one interrupt routine. When this happens the CPU basically has two ways of servicing the interrupts: the first is to run the routines sequentially, the second is to 'nest' the interrupt routines in order of priority.

In the former case the CPU first determines the source of the interrupt request then jumps to the appropriate routine. Whilst the CPU is executing this routine the interrupt input is inhibited, so that it will not respond to any further interrupts. Once the service routine has been completed the CPU resumes main programme execution. However, if the CPU is then presented with a second interrupt request, it will once more automatically branch to the required subroutine. The problem of several interrupt requests occuring whilst the CPU is already executing an interrupt routine is solved be means of a priority encoder which assigns a different priority to each interrupt source. The CPU must therefore be able to interrogate the encoder so as to determine the relative priority of the various interrupting devices. Figure 2a shows the sequence of subroutines; in this example routine 'ϕ' has the highest priority.



In the case of nested interrupts, the interrupt system is re-armed immediately upon the CPU branching to an interrupt routine, so that a second interrupt request can be acknowledged by the CPU at any time. Assuming, for example, that the CPU is already executing an interrupt routine when it detects a second interrupt request from a higher priority source, it will first branch to the routine which will service that device, then return to complete the initial interrupt routine, and only then return to the main programme (see figure 2b). During a routine the CPU will not acknowledge an interrupt request from a lower priority source.

Jumping from one programme to another does not, in itself, present any special problems; one must simply ensure that the contents of the various CPU registers are not lost when branching to a subroutine, otherwise the original programme could not be executed properly. To preserve the status of the CPU's internal register values, they are stored in a stack. This stack consists of several general-purpose registers which store data on the principle of last-in/first out ('lifo'). Some microprocessors possess an integrated stack register, whilst others have instructions* which permit a stack to be programmed into the RAM.

SC/MP interrupt system

The SC/MP has only one interrupt input (Sense A). When the internal interrupt enable (IE) flag is set, by executing either an Enable Interrupt Instruction (IEN) or a Copy Accumu-

* In several microprocessors one machine instruction results in the CPU carrying out a large number of separate steps. Think, for example, of how many operations are involved in a DLY instruction in the case of the SC/MP. The name for the total number of steps involved in a single machine instruction is a 'micro-programme'. Computers and some microprocessors are 'micro-programmable', i.e. the micro-programme, and so the instruction set, can be altered to suit a special application. Naturally this technique requires a profound knowledge of the architecture of the CPU in question.

1

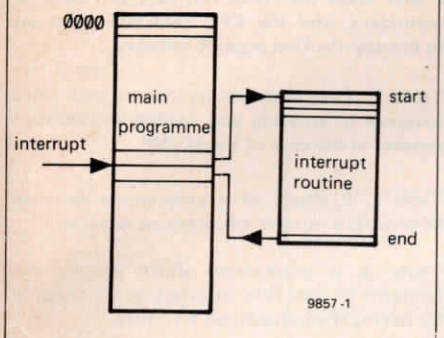
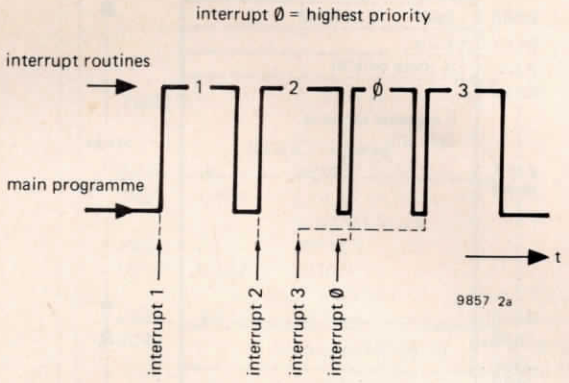


Figure 1. Diagrammatic representation of an interrupt operation.

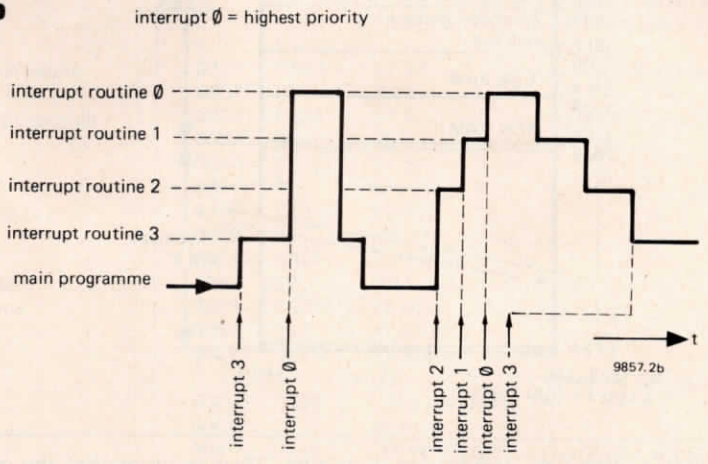
Figure 2. These examples illustrate the two basic methods of servicing multiple source interrupts.

Figure 3. Complete circuit diagram of all the hardware housed on the second Eurocard.

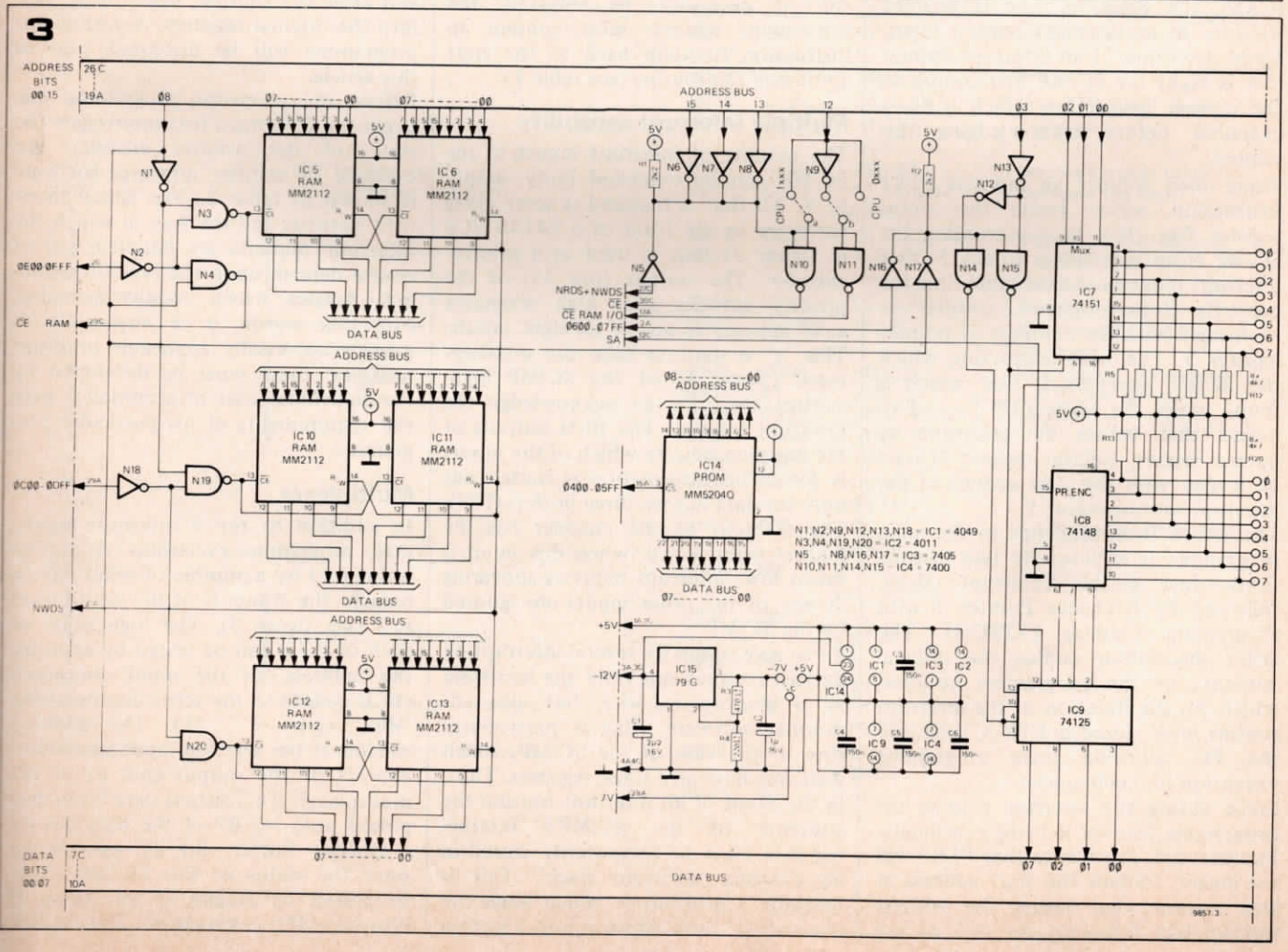
2a



2b



3



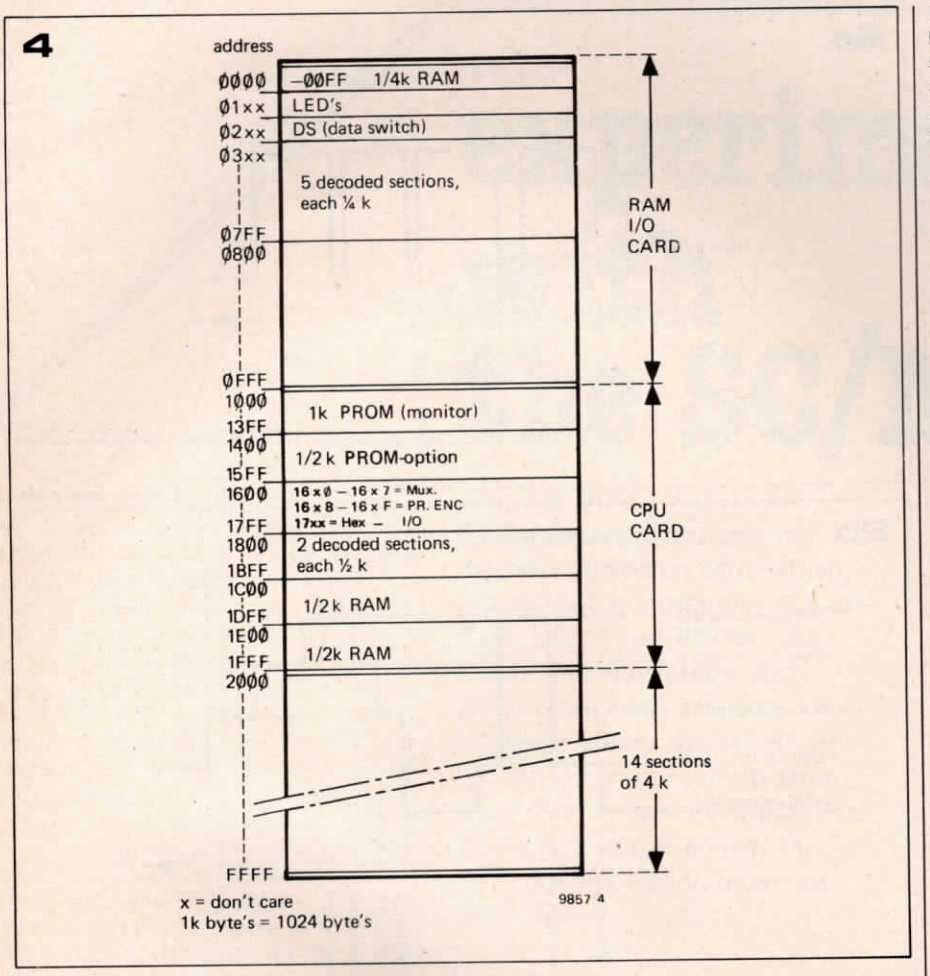


Figure 4. This diagram shows a detailed breakdown of the current page-address structure of the SC/MP system's memory. At a later stage the RAM I/O card will become redundant, and the CPU card will move up to occupy the first page of memory.

Table 1. This table shows the various steps involved in entering and exiting an interrupt routine in the case of the SC/MP.

Table 2. Example of a programme designed to process a number of interrupt requests.

Table 3. A programme which enables the contents of the CPU registers to be checked by having them displayed on LEDs.

Table 1

```
INT:    ; label of interrupt routine
 •
 •      } interrupt routine proper
 •
 •
IEN     ; enable interrupts
XPPC 3  ; return to main programme
JMP INT ; jump to start address of in-
          terrupt routine
```

lator to Status Register Instruction (CAS), the Sense A line is enabled to serve as an interrupt request input. Upon detection of an interrupt request (SA is high) the SC/MP first completes the current instruction which is being executed before acknowledging this request.

There then follows an internal DINT Instruction which resets the status register flag (IE), thus preventing the SC/MP from responding to any further interrupt requests. At the same time the contents of the programme counter are exchanged with the contents of pointer register 3. The next instruction which the SC/MP executes is that which is found under the address (PC) + 1. This means that before the interrupt sequence begins, pointer register 3 must be loaded with the start address of the interrupt routine minus 1.

The return from interrupt to the main programme is effected by two instructions: first Enable Interrupt (IEN), followed by Exchange Pointer 3 with Programme Counter (XPPC 3). The latter instruction copies the original contents of the programme counter, which for the duration of the interrupt routine were stored in PTR 3, back into the PC, allowing main programme execution to recommence.

Since during the interrupt routine the programme counter is being continually incremented, this means that PTR3 will no longer contain the start address of this routine, but rather the address immediately following the end of the routine. Thus in order that this routine can, if necessary, be repeated, the subsequent address must contain an instruction to jump back to the start address of the routine (see table 1).

Multiple interrupt capability

The number of interrupt inputs of the SC/MP can be extended fairly simply to 8. All that is required is some extra hardware in the form of a 74148 (IC8 in figure 3) that is used as a priority encoder. The output (pin 15) of the priority encoder goes high whenever a 'Ø' appears at one of its eight inputs. This '1' is used to take the interrupt input (Sense A) of the SC/MP high, causing the CPU to acknowledge the interrupt request. The BCD outputs of the encoder indicate which of the inputs is low. This information is routed out onto the data bus via three buffers (IC9). The Ø input of the encoder has the highest priority, i.e. when this input is taken low, interrupt requests appearing at any of the other inputs are ignored by the SC/MP.

To be able to service several interrupting devices requires not only the hardware of a priority encoder, but also additional software. This is particularly true in the case of the SC/MP, which does not have any stack registers. Thus, in the event of an interrupt routine the contents of the SC/MP's internal registers must be temporarily stored in an external 'software stack'. This is basically a programme which loads the contents of these registers into a section of memory reserved for this purpose, and after the routine, loads them back into the original registers. A part of this programme will be discussed later in this article.

Before the interrupt programme can begin the CPU must first interrogate the state of the priority encoder. An example of suitable interrupt software is shown in table 2. The actual interrupt routines and the way in which the interrupt requests are handled will of course depend upon the type of peripheral devices which require servicing. For this reason it is impossible to provide universally applicable interrupt routines, these must be developed by the individual user in accordance with the requirements of his particular programme.

Multiplexer

In addition to the 8 interrupt inputs, main programme execution can also be influenced by a number of other inputs, namely the 8 inputs of the multiplexer IC7 (see figure 3). The logic state of each of these can be tested by applying the 'address' of the input concerned (BCD-coded) to the select inputs of the 'Mux' (pins 9 . . . 11). The inverted version of the selected input signal then appears at the output (pin 6) of the multiplexer. This output data bit is then pulsed onto bit Ø7 of the data bus via a tri-state buffer. Bit Ø7 was chosen since the status of this bit can easily be tested by means of the Jump If Positive (JP) instruction. The SC/MP

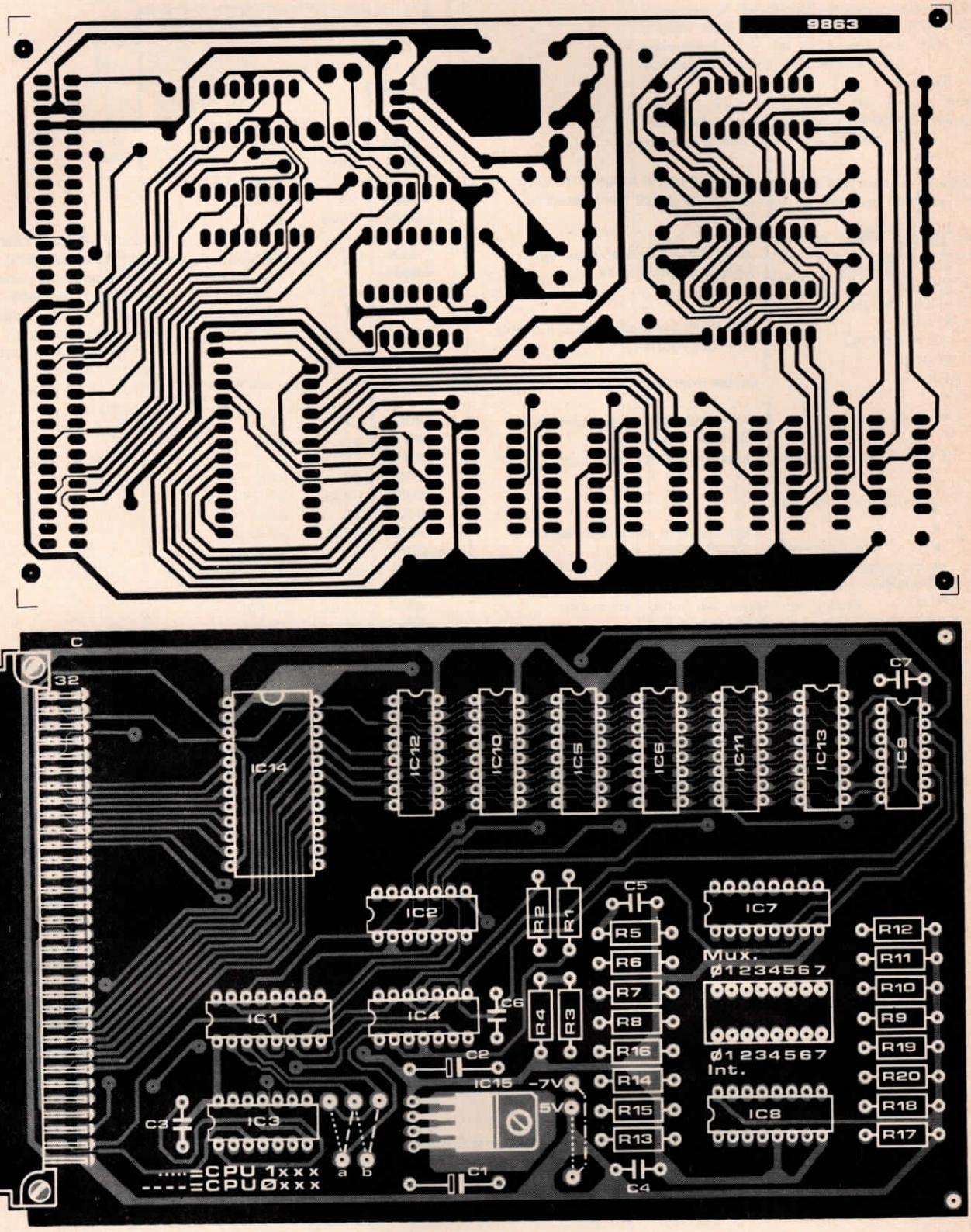
Table 2

```
MAIN PROG:
  DINT                  ; disable interrupts
    •
    •                   } section of main programme with
    •                   } SC/MP inhibited from detecting
    •                   } interrupts
LDI L (STACK)           }
XPAL2                   } load PTR 2 (stack pointer) with
LDI H (STACK)           } address of RAM stack
XPAH2                   }
LDI L (INTIN)–1         }
XPAL3                   } load PTR 3 with address of multiple
LDI H (INTIN)           } interrupt routine
XPAH3                   }
IEN                     ; enable interrupts
    •                   }
    •                   } section of main programme where
    •                   } interrupt is possible
INTIN:                  ; Label for multiple interrupt routine
    •
    •
    •
    •                   ; store status in stack (see table 3:
    •                     SAVSTA routine)
    •
LD PRIOR                ; interrogate priority encoder
ANI 07                  ; mask out number of routine
XAE                     ; and store in E
LDE
XRI 00                  ; was the number 0?
JZ INT 0                ; if yes, jump to INT 0
LDE
XRI 01                  ; was the number 1?
JZ INT 1                ; if yes, jump to INT 1
LDE
    •
    •
    •
  etc.
    •
    •
INT 0:                  ; label for routine '0'
 IEN                    ; only for nested interrupts
    •                   }
    •                   } routine '0'
    •                   }
    •                   }
JMP INTOUT              ; jump to INTOUT routine
INT 1:                  ; label for routine '1'
 IEN                    ; only for nested interrupts
    •                   }
    •                   } routine '1'
    •                   }
    •                   }
    •                   }
    •
  etc.
    •
    •
INTOUT:
    •                   }
    •                   }
    •                   } load status back into CPU
    •                   }
    •                   }
IEN                     ; only for sequential interrupts
XPPC3
JMP INTIN               ; only for sequential interrupts
```

Table 3

```
                START = 0000
0000  08        NOP
0001  C454      LDI L
                 (SAVSTA)–1
0003  33        XPAL3
0004  C400      LDI H
                 (SAVSTA)       ; load PTR 3 with the
0006  37        XPAH3            address of SAVSTA
0007                             ; the programme under test
                                   is loaded from 0007 on
      3F        XPPC3          ; load XPPC3 into the
                                 address immediately
                                 following the 'suspect'
                                 section of programme
                SAVSTA:        ; Label of SAVe STatus
                                 routine
0055  C837      ST AC          ; store (AC) in RAM
0057  01        XAE
0058  C835      ST E           ; copy (E) to RAM
005A  06        CSA            ; (SR), etc.
005B  C833      ST SR
005D  31        XPAL1
005E  C831      ST P1L
0060  35        XPAH1
0061  C82F      ST P1H
0063  32        XPAL2
0064  C82D      ST P2L
0066  36        XPAH2
0067  C82B      ST P2H
                MUX:           ; Label for multiplex routine
0069  C400      LDI L (MUX)
006B  31        XPAL1
006C  C416      LDI H (MUX)    ; load PTR 1 with the address
006E  35        XPAH1            of Mux
006F  C401      LDI H (LED)    ; load PTR 2 with the address
0071  36        XPAH2            of the LEDs
0072  C48D      LDI L (AC)
0074  33        XPAL3
0075  C400      LDI L (AC)     ; load PTR 3 with stack
0077  37        XPAH3            address
                LOOP:
0078  C407      LDI 07         ; prepare counter to in-
007A  C811      ST COUNT       ; terrogate the Mux
                NEXT:
007C  B80F      DLD COUNT      ; decrement counter and
007E  01        XAE              load E for indirect
                                 addressing
007F  C180      LD X'80 (1)    ; load Mux input 0 . . . 7
0081  94F9      JP NEXT        ; if addressed input active,
                                 continue
0083  C008      LD COUNT       ; load counter state into E
0085  01        XAE            ; and display desired
0086  C380      LD X'80 (3)    ; (register) contents on LEDs
0088  CA00      ST 0 (2)
008A  90EC      JMP LOOP
008C  00        COUNT:
                • BYTE
008D  00        AC:            ; RAM-byte for (AC) stack
                • BYTE
008E  00        E:             ; ditto for (E)
                • BYTE
008F  00        SR:
                • BYTE
0090  00        P1L:           ; etc.
                • BYTE
0091  00        P1H:
                • BYTE
0092  00        P2L:
                • BYTE
0093  00        P2H:
```

5



Parts list to figures 3 and 5

Resistors:
R1,R2 = 2k2
R3 = 470 Ω*
R4 = 220 Ω*
R5 . . . R20 = 4k7

Capacitors:
C1 = 2.2 μ/16 V
C2 = 1 μ/16 V*
C3 . . . C6 = 150 n

Semiconductors:
IC1 = 4049
IC2 = 4011
IC3 = 7405
IC4 = 7400
IC5,IC6,IC10 . . . IC13 = 2112
IC7 = 74151
IC8 = 74148
IC9 = 74125
IC14 = MM 5204Q
IC15 = 79G*

* omitted for SC/MP II

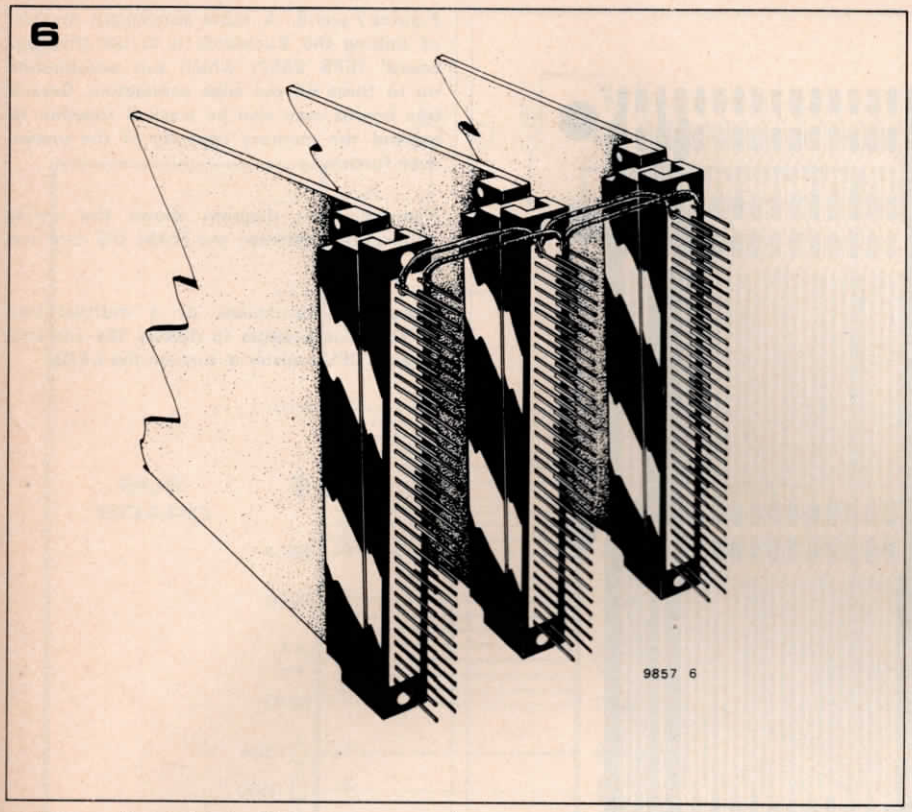


Figure 5. The track pattern and component layout of the printed circuit board for the memory card (EPS 9863). Particular attention should be payed to the wire link to the right of IC15: when using the earlier P-MOS SC/MP, this link should be in the '–7 V' position; for SC/MP II it should be in the '5 V' position.

Figure 6. To interconnect the Eurocards, a connector bus can be formed by joining the corresponding pins of the socket connectors using wire links.

will jump if this bit is 'Ø' (i.e. a positive number), and continue main programme execution if it is '1'. An example of the software involved when utilising the multiplexer is shown in the programme listed in table 3.

Page-address structure

As the volume of system hardware continues to grow with the addition of the memory card (shown in figure 3), so the need to clarify the address structure of the system becomes more urgent.

The CPU card already contains a large portion of the memory capacity of the system (e.g. the PROMs for the monitor software), which, naturally enough, must be capable of being addressed. The CPU card is therefore supplied with an address decoder. With the advent of the additional memory capacity represented by the circuit in figure 3 the page-address structure of the system's memory takes the form shown in figure 4.

Half of the first memory page (ØØØØ – ØFFF) can be addressed by the address decoder of the RAM I/O card. However, since the address decoding on the RAM I/O card is incomplete (AD 11 is not decoded) the second half of this page is identical to the first half and cannot therefore be used for additional hardware.

The second memory page contains everything which can be addressed by the address decoder of the CPU card. This consists firstly of the two PROMs for the monitor software. To provide the option of expanding the monitor software, space is provided for a third (½ k) PROM (IC14 in figure 3). The section from 16ØØ to 17FF is reserved for the multiplexer with priority encoder and for the hexadecimal input/output (HEX I/O) hardware which will be appearing shortly. The remaining lines of the second page are taken up by a 1 k RAM, ¼ k of which is present on the CPU card, with ¾ k situated on the memory card as shown in figure 3.

Once the hexadecimal input/output has been incorporated into the SC/MP system, the RAM I/O card will become largely redundant. Once the user has acquired a certain degree of proficiency with the system he can be expected to dispense with the RAM I/O card completely. For this reason it is also possible to construct the system without the RAM I/O card. This is done by switching the wire links a and b (shown as dotted lines in figure 3) to their alternative positions, so that the first memory page is now addressed by the address decoder of the CPU. Everything which in figure 4 lies between 1ØØØ and 1FFF is then situated between ØØØØ and ØFFF.

Board interconnections

The complete circuit diagram of the memory-extension card is shown in figure 3. The track pattern and component layout of the printed circuit board for this card are shown in figure 5. This board, like the CPU card, is double-sided with plated-through holes, and conforms to Eurocard dimensions. It should also be fitted with a 64-way edge connector. The board houses a regulator IC (IC15) which supplies the negative voltage for the earlier, PMOS version of the SC/MP (see Elektor 32, p. 12-08: SC/MP II and the p.c. board). If SC/MP II is used, IC15, R3, R4 and C2 can be omitted; the wire link adjacent to the position for this IC is then connected to the point marked '5 V'.

In order to interconnect the various Eurocards, a 'connector bus' is necessary. This basically is nothing more than a number of socket connectors with the corresponding pins (all pins 1a, all pins 1b, etc.) interconnected. This arrangement is illustrated in figure 6.

Whilst it is entirely possible to make all the necessary connections in this fashion (using e.g. 'wire-wrapped' links), such a method is both time-consuming and error-prone. For this reason a 'bus board', which can accomodate three socket connectors, was designed (see figures 7 and 8). The CPU card and the memory card can then be interconnected by simply plugging them into the bus board. This bus board must of course be able to communicate with the RAM I/O card. Since the RAM I/O card does not use edge connectors, these connections must be hardwired. The wiring details are provided by figure 9.

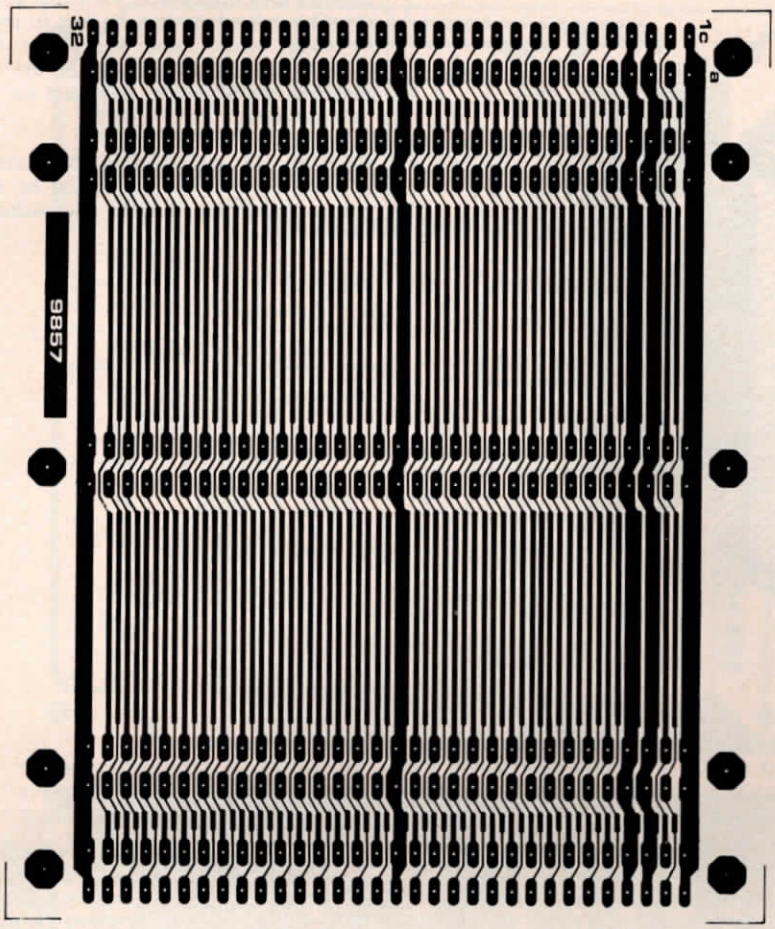
Termination points to the bus lines are provided at each end of the board so that several bus boards can be stacked end to end and linked to extend the system. The layout of these termination points allows the addition of extra 64-way sockets so that external connections need not be hard wired.

The only major limitation to large scale expansion of the system is that imposed by the power supply. Anyone who plans a large system should bear in mind that each page of memory (4 k) consumes a current of approximately 1 A. A suitable 5 V/3 A, –12 V/0.5 A supply will be published in the near future.

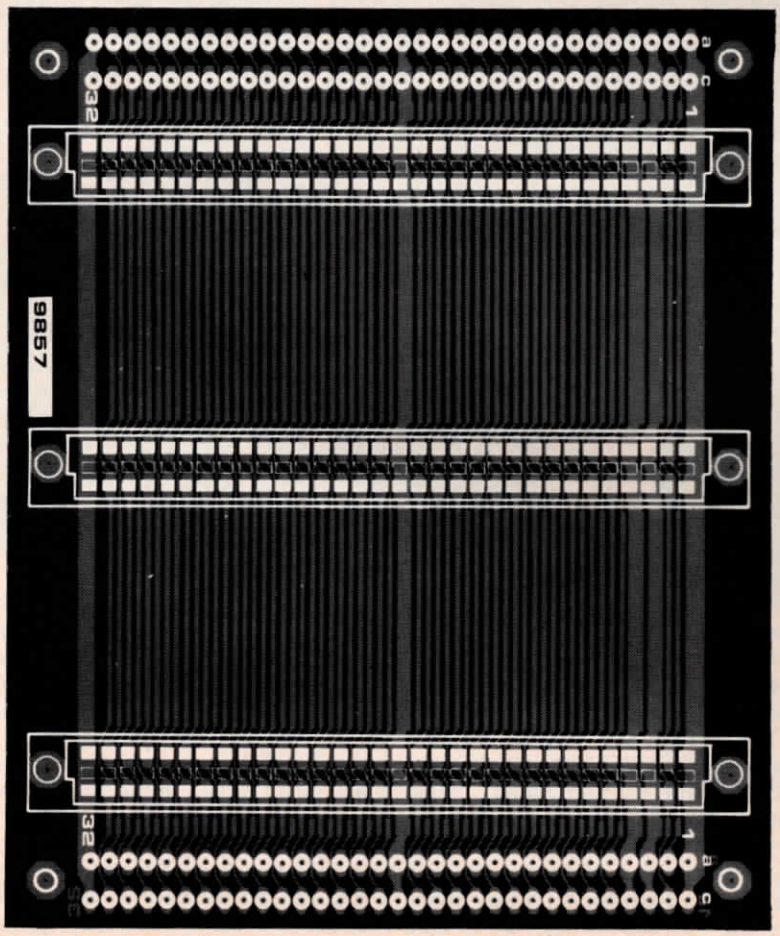
Software

It will not have escaped most readers that the role played by software in this series of articles is gradually growing in importance. The reason for this is twofold: firstly the 'intelligence' of a

7



8



Figures 7 and 8. A more convenient method of linking the Eurocards is to use this 'bus board' (EPS 9857) which can accomodate up to three 64 pin edge connectors. Several bus boards may also be stacked together to expand the memory capacity of the system even further.

Figure 9. This diagram shows the wiring connections between the RAM I/O card and the bus board.

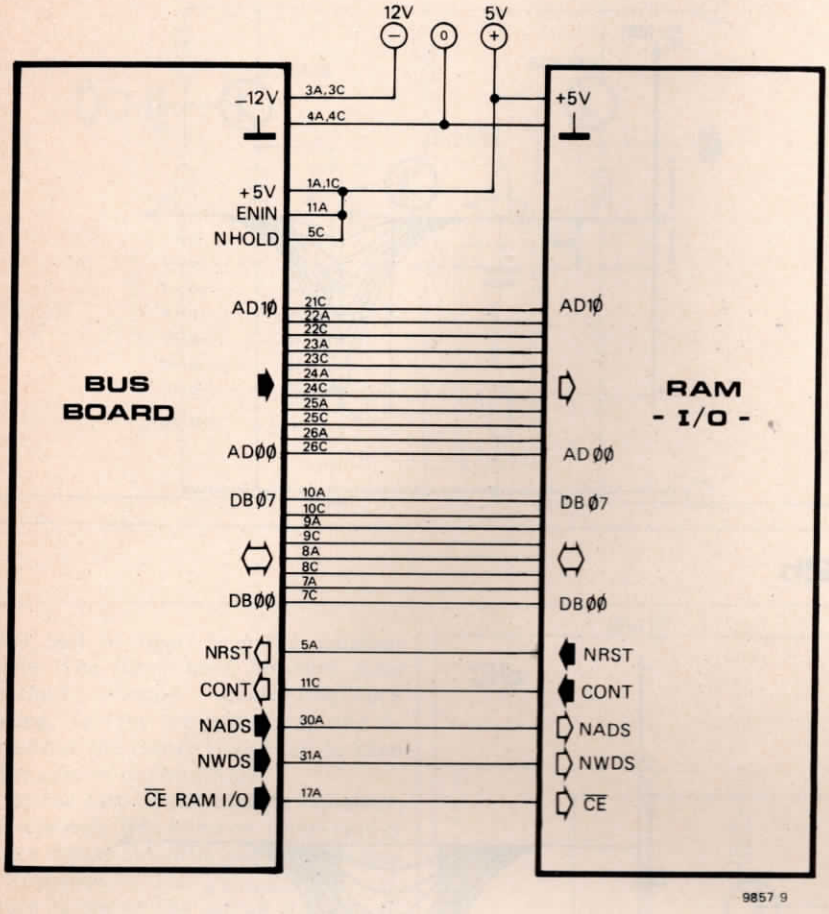
Figure 10. By means of a multiposition switch it is possible to display the contents of each CPU register in turn on the LEDs.

computer system is largely determined by the number and type of programme at its disposal; secondly, it is through developing his own software that the user can best appreciate the true potential of his system.

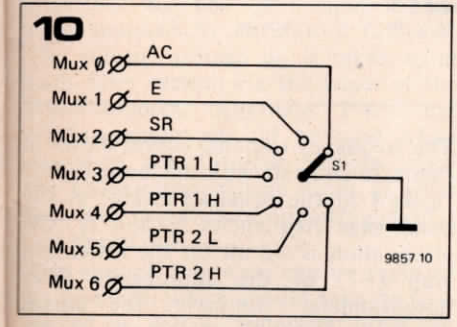
To this end, the present article concludes with a short 'debug' programme which will display the contents of the CPU registers at any stage during the programme under test. This is done by replacing the instruction which immediately follows the 'suspect' section of the programme by XPPC 3 (3F). The programme under test is then started as normal, after an NRST instruction. When the programme reaches the XPPC 3 it jumps to a 'save status routine' and writes the contents of the CPU registers into the RAM. The Mux inputs are connected to a multiposition switch (see figure 10), by means of which the register whose contents are to be displayed on the LEDs can be selected. In this way the contents of each CPU register, with the exception of course of PTR 3 and the PC, can be examined in turn.

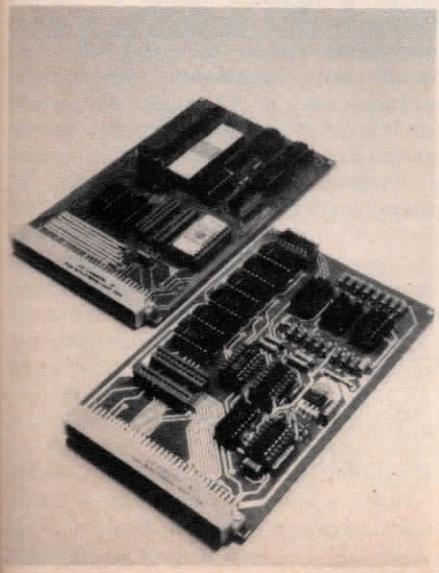
The programme in its present form can only be exited from by means of an NRST instruction. A more sophisticated and convenient version of the programme will later be incorporated into the monitor software.

9



10





The listing for this programme is given in table 3. The 'save status routine' can also be used for interrupt operations. In this case the section of RAM from ØØ8D . . . ØØ93 is used to form a software stack.

The majority of the instructions contained in this programme have already been discussed and require no further explanation. One important exception however is the 'indirect' address mode utilising the extension register. As explained in part 1 ('address modes'), indirect addressing describes the address mode whereby the effective address (EA) is obtained by incrementing the contents of a pointer by the contents of a byte taken from the RAM. In the case of the SC/MP this can only be done with the aid of the extension register. When the displacement value is X'8Ø, then (for memory reference instructions) it is no longer used to obtain the effective address, but is replaced by the contents of the extension register. The contents of the extension register are not known at the time of entering the programme, but are determined during execution of the programme. In this programme the indirect address mode results in a considerable saving in the number of instructions.

(to be continued)

missing link

Modifications to
Additions to
Improvements on
Corrections in
Circuits published in Elektor

Formant – the Elektor music synthesiser

Parts 4 and 5, October and November 1977. In Part 4, R1 in the input adder circuit (figure 7) is shown as 100 k. This gives the 'Octaves, coarse' control (P1) a range of ± 7½ octaves. At a later data, it was decided to reduce this range to ± 5 octaves. In part 5, this modification was carried out: in figure 2a, R1 is shown as 150 k; the front panel layout (figure 3) shows a 'coarse' range from –5 to +5.

However, we forgot to point out that the value of R1 shown in figure 7 of part 4 is 'incorrect'. Furthermore, and more seriously, the Octaves/Volt adjustment was based on the original value of R1. This means that *the Octaves/Volt adjustment procedure described in part 5 is incorrect.*

For the adjustment procedure using a DVM and a frequency counter, the connection between the slider of P1 and R1 must first be unsoldered. The free end of R1 is then connected to ground, and the slider of P1 is connected to the KOV input with S1 in position 'a'. Adjustment can now proceed as described in the original article.

The second adjustment procedure, using the beat note method, is correct as originally described.