Automatic register readout

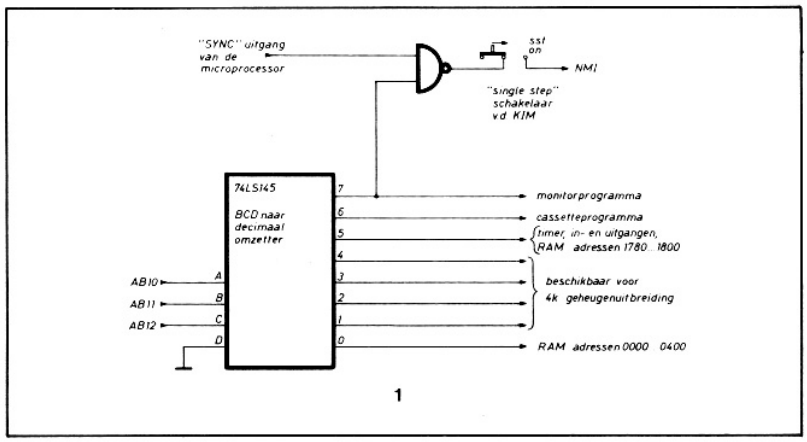
D. de Boer

When developing programs on the KIM 1, we can make use of the single-step facility that the KIM has. After each press of the 'GO' key, one instruction is executed.
The display then shows the address of the next instruction to be executed. After each step we can check the contents of the various registers. A drawback is that an address has to be entered for each register, which makes checking rather cumbersome. This article describes a program that ensures that, while the 'GO' key is held down, the contents of 3 memory locations of your choice appear on the display simultaneously. When the key is released, the address of the next instruction to be executed appears again.

How the single-step works

The single-step on the KIM makes use of the NMI (non-maskable interrupt). For it to work properly, it is therefore necessary to define the address to which the microprocessor must jump after an interrupt. For the single-step (and the ST key) this address is 1C00. At this address the monitor program begins by copying the internal registers into working memory. This allows us to check the contents of these registers and change them if necessary. After a press of 'GO', the (possibly modified) contents of the registers are put back and the microprocessor continues its original program. When the single-step switch is flipped, the 'SYNC' output is connected to the 'NMI' input of the microprocessor. The 'SYNC' output emits a pulse at the start of every instruction, so that an interrupt occurs during every instruction.
If we start a program with the single-step switch on, an interrupt will occur during the first instruction. As a result, the microprocessor jumps to the monitor program. Only when we press 'GO' again is the second instruction of the program begun. During this second instruction, however, another interrupt occurs, so that the program jumps to the monitor program again. In this way it is achieved that only one instruction is executed at a time. To get a proper single-step operation, however, something else is needed. In the story above we assumed that the interrupts occur exclusively during the instructions of the program under test. In reality (if no precautions were taken against it) an interrupt occurs during every instruction, so also during the instructions of the monitor program. The consequence would be that after the first interrupt, a jump is made to address 1C00, where the instruction 'STA' is located.
The microprocessor starts to execute this instruction, but during this instruction another interrupt occurs. The result is that the microprocessor jumps to address 1C00 again, with yet another interrupt. As a result the display will stay dark, and the stack is

completely filled up by the large number of interrupts. Therefore some way must be found to prevent interrupts from being generated while the monitor program is running. Fig. 1 shows how this problem has been solved on the KIM 1.



1. How the KIM's single-step works. When the monitor program is selected, no interrupts can occur.

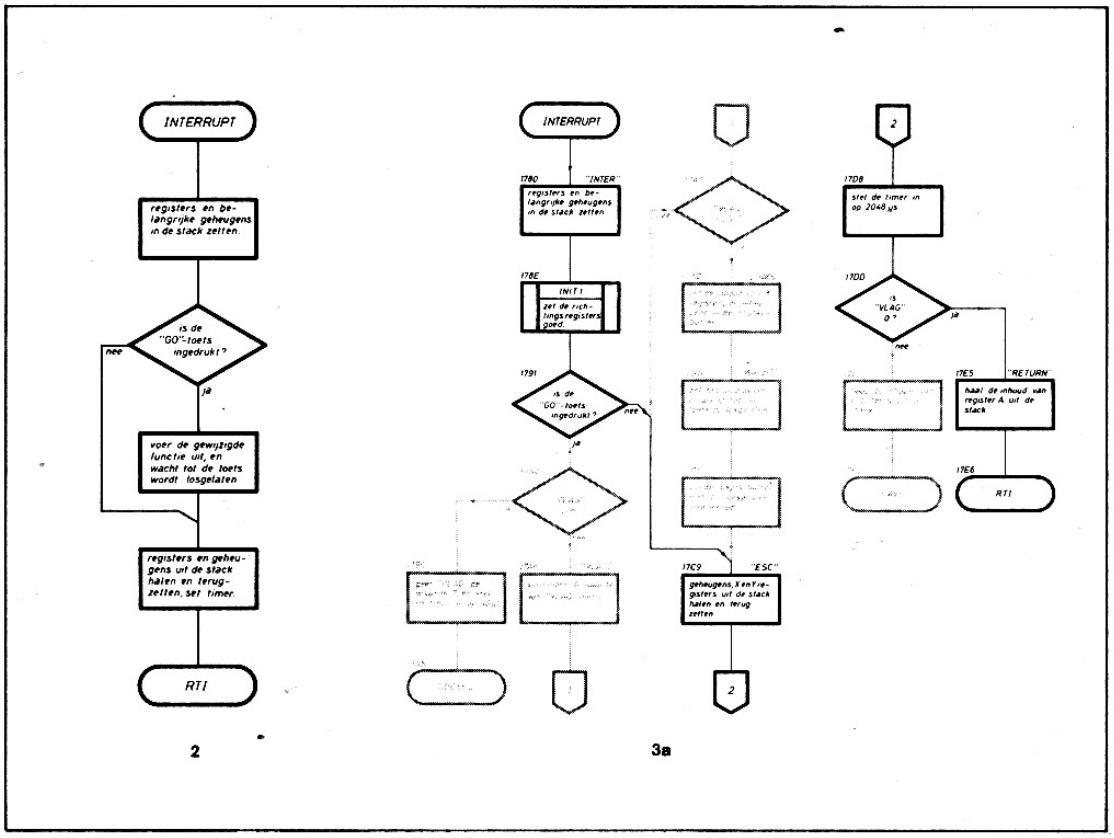
The KIM 1 contains a decoder for the first 8 K of address space. This decoding is realized with the 74LS145, a BCD-to-decimal converter. Output 0 selects the 1K RAM of the KIM; outputs 1, 2, 3 and 4 are available for 4K of memory expansion. Output 5 selects the RAM portion of the two 6530 ICs (with timer and inputs and outputs).
Output 6 selects the 1K cassette program, which is fixed in ROM. Finally, output 7 selects the monitor program. This means that output 7 goes low when the microprocessor selects an address in the 'monitor space'. This signal is now fed, together with the 'SYNC' signal, to a NAND gate. As long as output 7 is high (and therefore the monitor program is not selected) and the single-step switch is closed, the pulses from the 'SYNC' output will cause interrupts. When the monitor program is selected (after each interrupt), output 7 (fig. 1) goes low, so that the output of the NAND gate remains unconditionally high. No more interrupts occur and the microprocessor can now calmly execute the monitor program.
This immediately explains the phenomenon that the subroutines of the monitor program cannot be stepped through in single-step. (These subroutines are also quite often called in self-written programs.) When we jump to such a subroutine in single-step, the first interrupt will only occur again during the first instruction after the subroutine. As a result the program is only interrupted again at the second instruction after that subroutine. It is sometimes wrongly thought that the microprocessor jumps back to a wrong address, which is certainly not the case.

How can the single-step be modified?

Now that we know how the single-step is realized, we can start thinking about how to change its operation. We definitely do not want to start soldering and modifying on the KIM's circuit board. Firstly, because there is then a risk that the KIM is ruined for good, and secondly to prevent different KIM versions from arising. The alternative is to build a second single-step, in which the interrupt pulses are generated by a special program. This program must then be no larger than 103 bytes, so that it fits completely in the little-used RAM at addresses 1780 . . . 17E6.

The second single-step

Since the single-step on the KIM cannot be modified so easily, we will make a second single-step. Normally, when we press 'GO', a program is executed in its entirety. (The single-step switch is now off.) If, with a press of the 'GO' key, we also start the KIM's timer, we can make sure that an interrupt occurs during the first instruction of the program. In this way only one instruction will be executed. We are, however, faced with a small problem: the monitor program, which decodes which key has been pressed and executes the corresponding command, does not start a timer for the 'GO' key!!



2. The principle of changing the function of a certain key (here GO).

3a. As long as the 'GO' key is not pressed, no action is taken by the interrupt program.

It is therefore necessary to change the function of the 'GO' key. How this can be done is shown in fig. 2. We make sure that the monitor program is interrupted regularly by means of an interrupt. This short interrupt program then quickly checks whether the 'GO' key has been pressed. If this is not the case, we simply return to the monitor program. Here the timer ensures that interrupts are generated regularly. As soon as the 'GO' key is pressed, the interrupt program will (according to fig. 2) perform the modified function. Of course the monitor program must not take any action when 'GO' is pressed. Therefore the microprocessor may only leave the interrupt program once the key has been released. The monitor program will then no longer detect a key press, so that any earlier observations are undone. We do have to make sure that the interrupts follow each other quickly enough, because once the monitor program has started executing the 'GO' command, it will continue with it after the interrupt too.

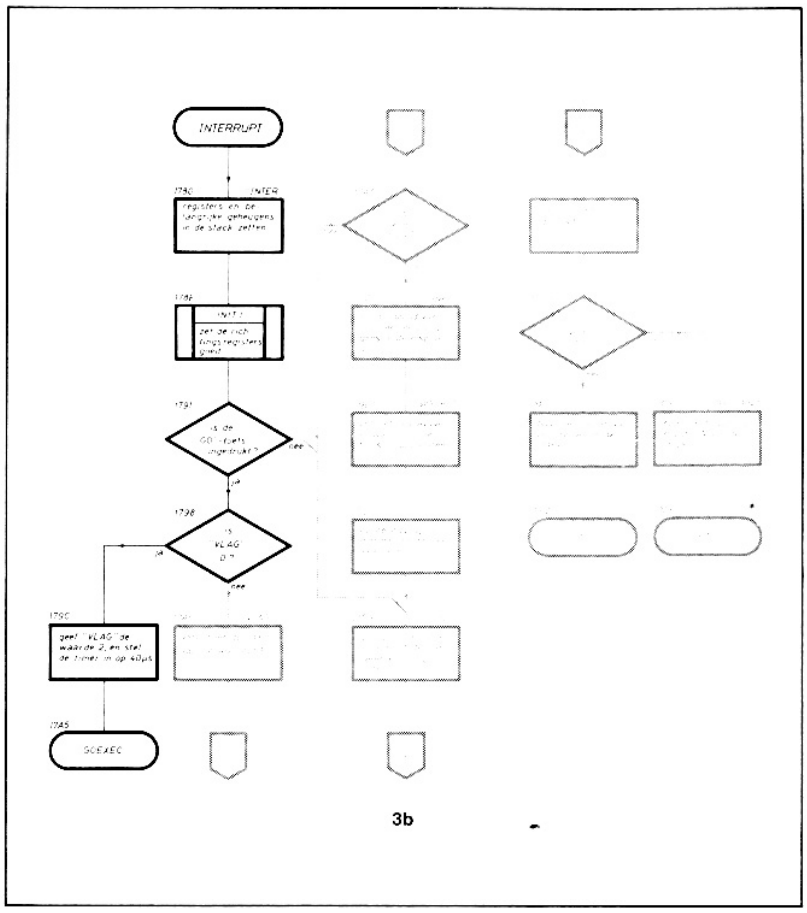
The interrupts

So we now use the timer to interrupt the monitor program regularly. To achieve this, PB7 must be connected to the NMI input of the KIM. The short interrupt program only checks whether the 'GO' key has been pressed and sets the timer to 2048 µ sec. As a result, after 2048 µ sec another interrupt will occur, so that the 'GO' key is looked at again. When the 'GO' key is pressed, only one command of the program under test must be executed. To achieve this, the timer must also be set, but now to 40 µs. This is the time needed to get from the interrupt program, via the monitor program, to the program under test. During the first instruction of this program an interrupt occurs and we now have to jump to the monitor program (address 1C00). This, however, goes via the interrupt program, because addresses 17FA and 17FB now contain the start address of the interrupt program.

The interrupt program therefore has to perform several tasks. Firstly, it must regularly interrupt the monitor program in order to check the 'GO' key. Secondly, when the 'GO' key has been pressed, it must start the program under test and immediately afterwards generate an interrupt. Thirdly, after this interrupt it must start the monitor program again. Of course the interrupt program must know which task it has to perform when an interrupt occurs. Therefore we keep track, in a memory location (00F6), of which task the interrupt program must perform. The following codes are used for this:

0 . . . . start the program under test
2 . . . . start the monitor program
1 . . . . wait until GO key is released.

The last code (wait until the GO key is released) is needed to prevent the monitor program from executing the GO command once more.



3b. When the 'GO' key is pressed, the timer is set to 40 µs, and we jump via 'GOEXEC' to the program under test. 'VLAG' was 0, and becomes 2.

The flow chart of the interrupt program

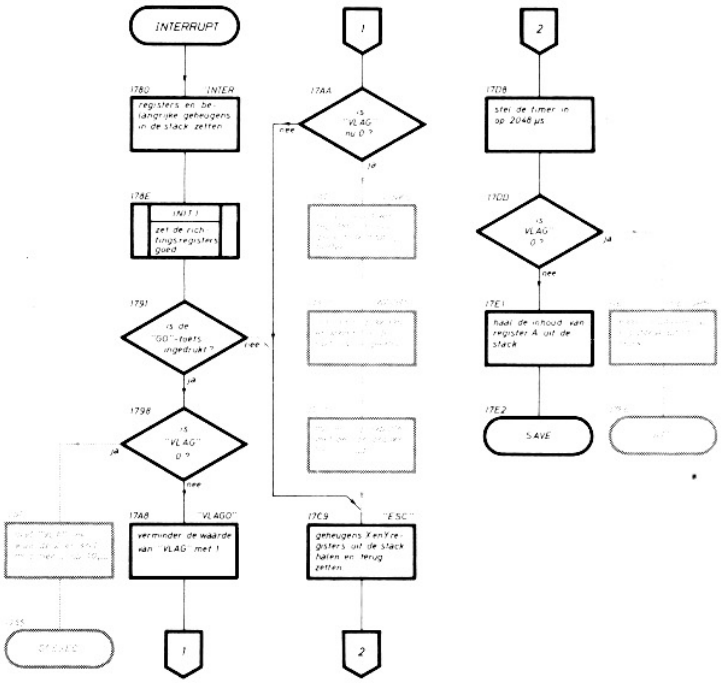
Fig. 3 shows the flow chart of the interrupt program. Immediately after the interrupt, care must first be taken that all register contents are preserved. Therefore registers A, X and Y are put on the stack, one after the other. The contents of memory locations 1740 . . . 1743 must also be temporarily stored elsewhere. These memory locations are used by both the monitor program and the interrupt program for the keyboard and display. When all data has been saved, we must set the direction register at address 1741 correctly. For this we jump to the subroutine 'INIT' of the monitor program. Everything is now in order to see whether a key has been pressed. To determine this we jump to the subroutine 'GETKEY'. Only when the GO key has been pressed does register A get the value $ 13.

If the GO key has not been pressed (fig. 3a), the registers are refilled with their original contents, the timer is set and via RTI we continue with the monitor program as if nothing had happened.

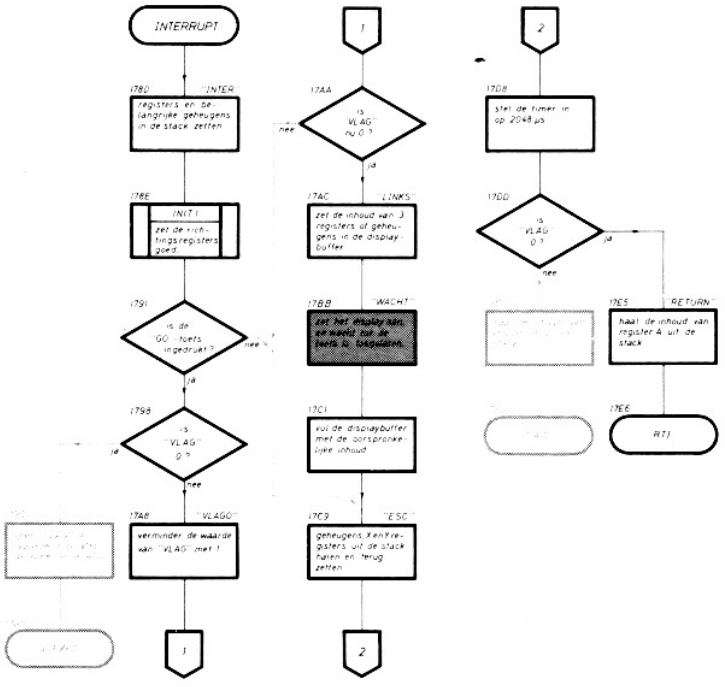
The timer then ensures that after 2.048 ms another interrupt occurs. The contents of 'VLAG' (address 00F6) now have the value 0. As soon as the GO key is pressed (fig. 3b),

we go via addresses 1794 and 1798 to address 179C. Here 'VLAG' is set to 2, so that at a following interrupt this route is not taken again.
Next the timer is set to $28 µs (40 µs decimal) and via a JMP instruction the microprocessor arrives at 'GOEXEC', address IDC8. The program section 'GOEXEC' ensures that the registers of the microprocessor are refilled with the values that belong to the program under test. These values had previously been placed at addresses 00EF . . . 00F5 by the monitor program. The stack pointer is also put back in the correct position. Via an RTI the microprocessor now jumps to the program under test. Immediately the timer causes an interrupt and the interrupt program is started again. Now, however, 'VLAG' has the value '2'.
The interrupt program now runs as follows (fig. 3c). Via addresses 1780, 178E, 1791, 1794, 1798 (the GO key is still pressed) to 17A8 ('VLAG' = 2). At this address the contents of 'VLAG' are decreased by 1, and thus become '01'.
Doing this has a number of advantages; firstly, 'VLAG' has now received its new value with a short instruction, and secondly, shortly we can give 'VLAG' the value '01' once more with the same instruction, and we thus go via address 17AA to address 17C9. At this address the restoring of the registers and memory contents begins. We ultimately have to jump to the start of the monitor program, and the contents of memory locations 1740 . . . 1743 are actually of no importance. However, we have to run through this part in order to put the stack pointer back in its original position. Of course there are other possibilities for this, such as 'LDX and TXS' or simply 10xPLA. The method we follow here, however, requires the least memory space, because the program from address 17C9 onwards was needed anyway for the case that the 'GO' key has not been pressed. At address 17DD we check the contents of 'VLAG' to determine how we must leave the interrupt program. In our case 'VLAG' has the value 01, so that via addresses 17E1 and 17E2 we jump to 'SAVE'.
Because we use register A when determining the contents of 'VLAG', we put back the original contents of register A only at the last moment. 'SAVE' is the start of the monitor program; here the internal registers of the microprocessor, as they were when the program under test was left, are placed at addresses 00EF . . . 00F5. After 2048 µs another interrupt occurs and the GO key is still pressed. We now arrive via addresses 1780, 178E, 1791, 1794, 1798 at 17A8. Here 'VLAG' is decreased by 1 again, so that its contents are now '00'. As a result we go via 17AA to 17AC. We have now executed one instruction of the program under test and we may only leave the interrupt program when the 'GO' key is released. (Otherwise more instructions will be executed.) The address space 17AC . . . 17C0 is now free to use for putting certain values on the display. In the program we have put the contents of register A on the left 2 displays (contents 00F3 to 00FB), the contents of register X on the middle 2 displays (contents 00F5 to 00FA) and the contents of register Y on the right 2 displays (contents of 00F4 to 00F9).
Of course, while checking a program, other important memory locations can also be examined. To keep the whole thing flexible, no zero page addressing has been used, so

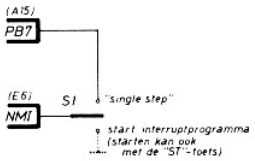
that addresses from anywhere in memory can easily be typed in. At address 17BB we jump to the display subroutine, which activates the displays. When the microprocessor returns from this subroutine, the contents of register A are not equal to '00' as long as a key is pressed. At address 17BE, as long as a key is pressed, a jump is therefore made back to address 17BB, where the display subroutine is run through again. As soon as the key is released, the display buffer is refilled at address 17C1 with the original contents (the address of the next instruction to be executed). Next, from address 17C9 the contents of the registers and memory locations are put back. 'VLAG' is now '00' and we therefore leave the interrupt program via 17DD, 17E5 and 17E6.
In summary: the 'GO' key is not pressed and the monitor program is running, so that the start address of the program under test is on the display. The monitor program is regularly interrupted by the interrupt program. The memory location 'VLAG' has the value '00' (fig. 3a). As soon as the 'GO' key is pressed, after the next interrupt the interrupt program will set 'VLAG' to '02' and, regardless of the place where the monitor program was broken off, jump to 'GOEXEC' address 1DC8 of the monitor program (fig. 3b). This piece of the monitor program starts the program under test. At the first instruction of this program an interrupt occurs, and we go back to the interrupt program. Now 'VLAG' is set to '01' and the monitor program is started at 'SAVE' (fig. 3c). At the next interrupt the interrupt program gives the memory location 'VLAG' the value '00' and waits until the GO key is released. During this time arbitrary memory locations can be made visible (fig. 3d). When the 'GO' key is released, the monitor program is continued where it was interrupted. From this moment the whole process starts over again.


INTERRUPT
1780 "INTER"
registers en belangrijke geheugens in de stack zetten
178E
INIT1
zet de richtingsregisters goed
1791
is de "GO"-toets ingedrukt?
nee
ja
1798
is "VLAG" 0?
ja
nee
17A8 "VLAG0"
vermindert de waarde van "VLAG" met 1
1
17AA
is "VLAG" nu 0?
nee
ja
17C9 "ESC"
geheugens X en Y registers uit de stack halen en terug zetten
2
1
2
17D8
stel de timer in op 2048 µs
17DD
is "VLAG" 0?
ja
nee
17E1
haal de inhoud van register A uit de stack
17E2
SAVE

3c


INTERRUPT
1780 "INTER"
registers en belangrijke geheugens in de stack zetten
178E
INIT1
zet de richtingsregisters goed
1791
is de "GO"-toets ingedrukt?
nee
ja
1798
is "VLAG" 0?
ja
nee
17A8 "VLAG0"
vermindert de waarde van "VLAG" met 1
1
17AA
is "VLAG" nu 0?
nee
ja
17AC "LINKS"
zet de inhoud van 3 registers uit geheugens in de displaybuffer
17BB "WACHT"
zet het display aan, en wacht tot de toets is losgelaten
17C1
vul de displaybuffer met de oorspronkelijke inhoud
17C9 "ESC"
geheugens X en Y registers uit de stack halen en terug zetten
2
1
2
17D8
stel de timer in op 2048 µs
17DD
is "VLAG" 0?
ja
nee
17E5 "RETURN"
haal de inhoud van register A uit de stack
17E6
RTI

3d


(A15)
PB7
(E6)
NMI
S1
"single step"
start interruptprogramma
(starten kan ook met de "ST"-toets)

4

3c. After the first instruction of the program under test we jump to the interrupt program again. Now we go to 'SAVE', where the internal registers of the microprocessor are saved. 'VLAG' was 2, and becomes 1.

3d. The last cycle. When all registers have been saved, the monitor program is interrupted again. Now the interrupt program keeps waiting until the 'GO' key is released. 'VLAG' was 1, and becomes 0.

4. The connection of the new single-step switch.

Starting the interrupts

The interrupt program itself keeps generating the interrupts. Once, however, the interrupts have to be started. At first sight one would say that a press of the 'ST' key is sufficient, since with 'ST' we do give an NMI. A press of 'ST' does indeed cause a jump to the interrupt program and thus the timer to be started. If, however, after 2048 µs the timer gives an interrupt, the 'ST' key is still pressed and so the NMI is still low. As a result the timer's interrupt will be lost and there will be no new jump to the interrupt program. Therefore switch S1 (fig. 4), the new single-step switch, must be in the 'off' position. When we now press 'ST', the interrupt program will indeed start the timer, so that PB7 goes low. If we now first release the ST key and then set S1 to single-step, an interrupt will occur again (PB7 was low). With this interrupt the interrupt program will be started again, so that from now on the interrupts are generated automatically. (The process can also be started by simply typing in address 170E.)

Further possibilities

By making changes between memory locations 17AC and 17C9 we can, for example, have a program stop carried out automatically every second. It would also be possible to check the program stop address and stop only at certain memory locations. (This is handy with loops.) Another handy function is a '–' key, which performs the opposite function of the '+' key.
We could use the 'PC' key for this. All these functions seem equally attractive, but would use too much memory for the basic version of the KIM. After all, the interrupt program is an auxiliary program for developing much larger programs in the remaining memory. Nevertheless it can be interesting to try to realize these possibilities.
The interrupt program as published here can be placed anywhere in memory, provided the start address is set at 17FA and 17FB.

Lijst 1

```
1780  48          INTER   PHA-impl
1781  8A                  TXA-impl
1782  48                  PHA-impl      Zet de inhoud van de
1783  98                  TYA-impl      registers in de stack.
1784  48                  PHA-impl
1785  A2 04               LDX-imm   $04
1787  BD 3F 17    SAVE    LDA-abs, X $173F
178A  48                  PHA-impl      De inhoud van 4
178B  CA                  DEX-impl      geheugens in de stack.
178C  D0 F9               BNE-rel   SAVE
178E  20 8C 1E            ISR-abs   INIT1     richtingreg. goed zetten.
1791  20 6A 1F            ISR-abs   GETKEY    Is 'GO' ingedrukt?
1794  C9 13               CMP-imm   $13       Zo nee, naar ESC.
1796  D0 31               BNE-rel   ESC
1798  A5 F6               LDA-Z page VLAG     Indien VLAG=0
179A  D0 0C               BNE-rel   VLAG0     naar VLAG0
179C  A9 02               LDA-imm   $02       VLAG op 2
179E  85 F6               STA-Z page VLAG
17A0  A9 28               LDA-imm   $28       Timer op 40 µs
17A2  8D 0C 17            STA-abs   TIM1T     (28 hex).
17A5  4C C8 1D            JMP-abs   GOEXEC    Naar monitor.
17A8  C6 F6       VLAG0   DEC-Z page VLAG     Verminder VLAG,
17AA  D0 1D               BNE-rel   ESC       bij 0 naar ESC.
17AC  AD F3 00    LINKS   LDA-abs   ACC       Reg. A op de 2
17AF  85 FB               STA-Z page POINTH   linker displays.
17B1  AD F5 00    MIDDEN  LDA-abs   XREG      Reg. X op de 2
17B4  85 FA               STA-Z page POINTL   middelste displays.
17B6  AD F4 00    RECHTS  LDA-abs   YREG      Reg. Y op de 2
17B9  85 F9               STA-Z page INH       rechter displays.
17BB  20 1F 1F    WACHT   JSR-abs   SCANDS     Display aan, en
17BE  D0 FB               BNE-rel   WACHT     wacht tot GO
17C0  EA                  NOP-impl            is losgelaten.
17C1  A5 EF               LDA-Z page PCL
17C3  85 FA               STA-Z page POINTL   Vul de displaybuffer
17C5  A5 F0               LDA-Z page PCH      met de oorspr. inhoud.
17C7  85 FB               STA-Z page POINTH
17C9  A2 00       ESC     LDX-imm   $00
17CB  68          TERUG   PLA-impl
17CC  9D 40 17            STA-abs, X $1740    Inhoud van 4 geheugens
17CF  E8                  DEX-impl            uit de stack.
17D0  E0 04               CPX-imm   $04
17D2  D0 F7               BNE-rel   TERUG
17D4  68                  PLA-impl
17D5  A8                  TAY-impl            Inhoud van X en Y
17D6  68                  PLA-impl            uit de stack.
17D7  AA                  TAX-impl
17D8  A9 02               LDA-imm   $02       Timer op 2048 µs.
17DA  8D 0F 17            STA-abs   TIMKT
17DD  A5 F6               LDA-Z page VLAG     Indien VLAG=0
17DF  F0 04               BEQ-rel   RETURN    naar RTI.
17E1  68                  PLA-impl            A uit de stack
17E2  4C 00 1C            JMP-abs   SAVE      naar monitor.
17E5  68          RETURN  PLA-impl            A uit de stack naar het
17E6  40                  RTI-impl            onderbroken programma.
```

Listing 1. The interrupt program. The numbers in bold are the addresses of the memory locations that are made visible and can be changed as required.

Instructions for use

To use the program we must perform the following actions.

a. one-time:

1. A switch must be soldered between PB7 and NMI as shown in fig. 4.
2. The program must be typed in at addresses 1780 . . . 17E6 or, if this memory space is used for another program, at any other place in memory (of course the program must not overlap the reserved areas of memory).
3. The NMI vector must be entered. If the program is typed in starting at address 1780, we put '80' at address 17FA and '17' at address 17FB.
4. Memory location 'VLAG' must be set correctly ('00' at address 00F6).

b. operation:

Switch S1 must be in the 'Normal' position. We press 'ST' once and the program is started. As long as S1 is in the 'normal' position we notice nothing of this. As soon as S1 is set to 'Single step', one instruction will be executed with every press of 'GO', while during the key press the contents of registers A, X and Y respectively become visible. Other registers and/or memory locations can also be made visible with a small modification.

Note!!

When using this second 'single-step', the single-step switch of the 'KIM' must be off!! The 'ST' key does not work; stopping is only possible with the 'RS' key.