



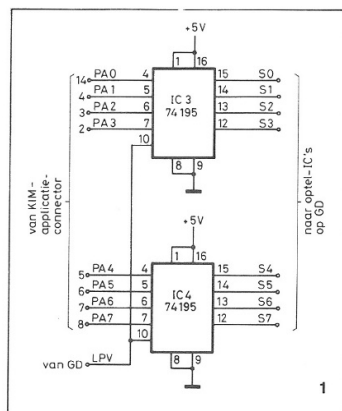
Graphic Display

Monitor for the KIM

*M. Dohmen,
R. Koekoek*

In the coming months we want to discuss a number of routines that will eventually lead to a complete monitor for the KIM using the graphic display (GD). To achieve this, some hardware modifications have to be made. You can expect a number of routines for, among other things, drawing lines and figures, ranging from simple squares and cubes to complicated ellipses, parabolas and hyperbolas. All this and much more works by means of a completely different programming language, a kind of "drawing language". Furthermore a complete machine-language monitor with disassembler and full cursor control, and a new cassette read/write routine with a storage speed of 2400 baud (that is 15.6K per minute) with software control of the recorders. Naturally this cassette routine comes with a "linking loader", which is universally applicable and allows extensions.

The set-up is as follows. The complete package consists largely of subroutines that are controlled by a small main program. These routines are simply placed one after the other in memory. Those who do not want a complete monitor can use these subroutines separately, since they can also prove their worth on their own. This produces a "subroutine arsenal". Those who do want a complete machine-language monitor, however, will have to make a few changes to, and additions to, the hardware.



1. As you may know, it is possible to build a hardware scroll-up on the GD. This needs 4 ICs. Without this extension the monitor does not work.
2. The monitor needs a larger keyboard (28 keys). I may assume, however, that if you do not have an ASCII keyboard you have long been planning to buy a more extensive keyboard. Further on in this article a keyboard is described that is very cheap and more than adequate (for about f 30.00 you have a well-working keyboard). A real ASCII keyboard is of course still recommended.
3. Not necessary, but very handy, is a circuit that allows a

Diagram illustrating the Synchprint van GD system architecture. The system is centered around an **optel-IC's** (Optel Integrated Circuit) block, which is connected to seven input lines labeled **S0** through **S7**.

The system includes several memory modules and logic components:

- Memory Modules:**
 - Five **74193** modules arranged in a vertical column.
 - One **74105** module.
 - One **7474** module.
- Logic Components:**
 - Two **7400** modules.
 - One **7404** module.
 - Two **7420** modules.
 - Two **7483** modules.

The connections show a complex interplay between these components, with the **optel-IC's** block acting as the central processing unit, receiving data from the input lines and interacting with the memory and logic modules.

Until now the input port of the GD can process four commands, namely:

001₂ = clock in x address.

011₂ = write a dot on the screen.

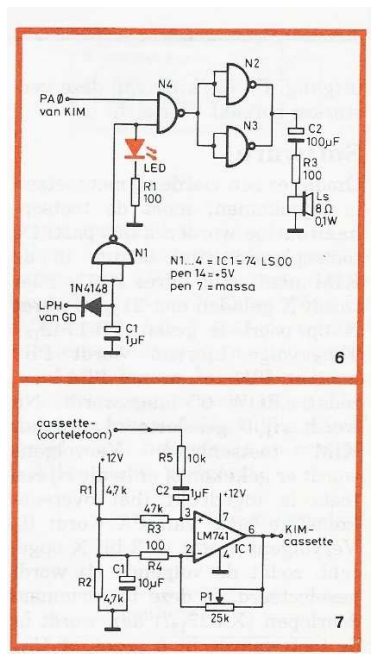
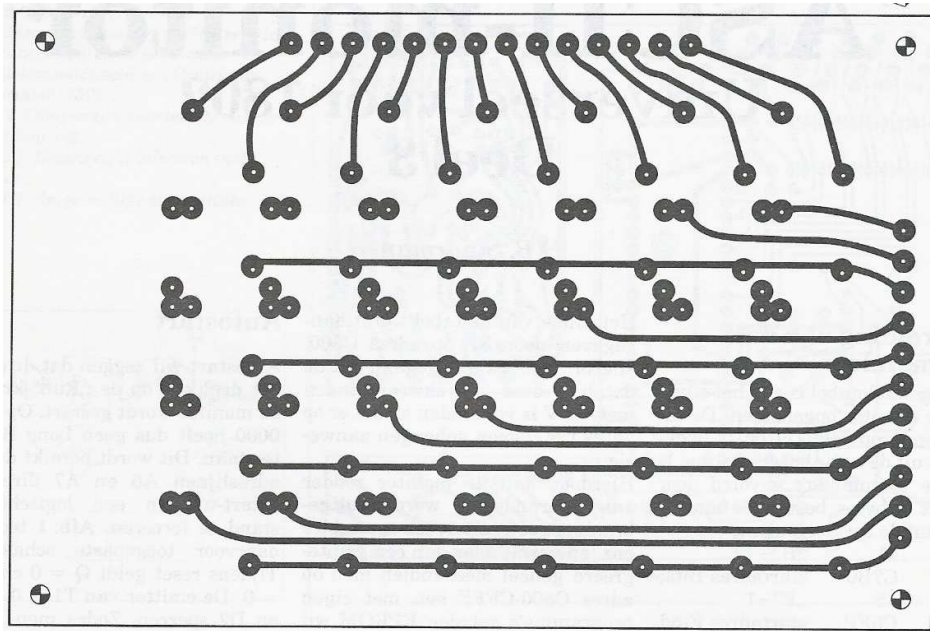
However, seven signals can be detected. The scroll-up uses the light-pen detection (101_2). As soon as 101_2 appears on the input port, LPV (light pen vertical) goes low and ICs 3 and 4 become active (see fig. 1). These two ICs are 4-bit shift registers. Their inputs are connected to port

[illegible]

For the extended monitor you need a keyboard with at least 28 keys. An ASCII keyboard is of course ideal, but because not everyone wants to spend the money on one, a cheap alternative is discussed here. On the KIM one pin of the binary-to-decimal decoder IC has been left unused. By

Fig. 3 shows part of the circuit around the KIM keyboard and the added hardware. As you can see, the new keys get the key codes 15_{16} to $1B_{16}$. It is a bit awkward to operate seven separate keys in addition to the original KIM keyboard, which will probably be some

distance away from the KIM keyboard. By mounting 28 new keys on one PCB and connecting it to the KIM, you have all keys together. If you do this, it is advisable to mount seven extra keys on the PCB and also to leave enough space for further extensions. If you buy good keys, you will never have to replace the keyboard. You can now also give the keys your own legends. What these legends should be will follow later. Fig. 4 shows the PCB layout of the keyboard. If you do not have the means to etch a PCB, you can fit keys to a pre-drilled board, which is just as handy.



Loudspeaker modification

Fig. 6 shows the complete circuit of the loudspeaker modification. This modification uses the LPH pin (light pen horizontal). Normally this output is high, but as soon as 110₁₂ [sic; 110₂] appears on the input port of the graphic display, a square wave with a frequency of 1 MHz appears on this output. This allows capacitor C1 to discharge, and the output of gate N1 goes high. The LED lights up to indicate that the loudspeaker is connected to PA0. The signal from PA0 is fed via N4 to the amplifier formed by N2 and 3, and then to a loudspeaker. An extra modification concerns the amplifier that must amplify the weak signal from the cassette recorder so that the signal can be fed to the KIM (see fig. 7). This modification is of course not necessary, but it is recommended if you want to be rid for good of all that "fiddling about" with the

volume control, or if you have had enough of the noise coming from the loudspeaker of your cheap cassette recorder. The input of the amplifier is now simply connected to the line output. The operation of this amplifier speaks for itself.

Software

Because a fourth row of keys has been added, the key input routine must be modified. The key input routine in the KIM monitor starts at address 1F6A. Here X is loaded with 21_{16} and X is put on port B (PB: 1742_{16}). As a result PB0 goes high and PB1 to PB4 go low, so that "ROW 0" goes low. Row 0 of the KIM keyboard is now selected. Next it is checked whether a key in that row is pressed (the corresponding bit of PA becomes 0). Then 2 is added to X, so that the next row is selected. When this loop has been run three times ($X=27_{16}?$), either the key value is put in A, or 15_{16} is put in A if no key was pressed. We

Lijst 1

```

ASS-6502 COMPUTER BULLETIN DB PAGE 01

0000:
0010:
0020:
0030:
0040:
0050:
0060:
0070:
0080:
0090:
0100:
0110: E000
0120:
0130:
0140:
0150:
0160: E000
0170: E000
0180:
0190: E000 A2 21
0200: E002 A0 01
0210: E004 20 02 1F
0220: E007 00 07
0230: E009 E0 29
0240: E00B D0 F5
0250: E00D A9 FF
0260: E00F 40
0270: E010 4C 7H 1F

*****
TOETSOPHAALROUTINE
MONITOR VOOR HET
GRAFISCH DISPLAY
M. DOEMEN EN R. FOKKOEK
11-80 RS
*****

ORG #E000

ROUTINES UIT DE MONITOR
VAN DE KIM

ONEKEY * #1F02
KEYIN * #1F7H

TOETS LOVIN #21 EERSTE RIJ IN REG X
MOTO LOVIN #01
JSA ONEKEY TOETS INGEPUKT IN DEZE RIJ?
BNE ZETOM DO JA, ZET WAARDE OM
CP#IM #29 VIER RIJEN GEHAFT?
BNE MOTO DO NEE, VOLGENDE
LOVIN #FF DO JA, ZET #FF IN ACCU
RTS EN KIEP TERUG
ZETOM JMP KEYIN ZET TOETSWAARDE OM

```

need the same routine for the new keyboard, with the change that we must select four rows instead of three. Instead of comparing register X with 27_{16} , we must compare register X with 29_{16} . We made a second change as well. If no key was pressed in the old routine, 15_{16} ended up in the accumulator. However, 15_{16} has become the code of a new key, so we must put a different value in A. If no key is pressed now, FF_{16} ends up in A.

This also has the advantage that it is very easy to check whether a key is pressed, namely with the BPL instruction (branch if the number in the accumulator is positive). Listing 1 shows the new key input routine. What you will probably notice when you study the routine is that the subroutine ends with an absolute jump to the KIM monitor. Well, in the KIM monitor the routine that is jumped to does end with an RTS instruction.

Captions part 1:

Fig. 1 Modification for the hardware scroll-up.

Fig. 2 Place on the sync board where the ICs and the wires of the modification must go (two 7483s).

Fig. 3 Keyboard and part of the KIM hardware.

Fig. 4 PCB layout of the new keyboard.

Fig. 5 Component layout of the keyboard.

Fig. 6 Loudspeaker connection.

Fig. 7 Amplifier for the cassette recorder.

Listing 1 New key input routine.

In the first part (Dec. '81) we discussed the various hardware modifications needed to make our monitor work on the KIM. In this part we discuss the software as it can be placed in the first EPROM and part of the second. We deal with the control of the scrolling, the loudspeaker and the cursor, which blinks at a variable rate. Furthermore the commands "clear the screen", "clear from the cursor to the end of the screen", "clear from the cursor to the end of the line", and print a pointer, byte, nibble or ASCII character. In the end the monitor needs about 6K of memory, which is very little given its many possibilities.

If you have difficulties connecting EPROMs, I advise you to read the article "Motherboard for 6502" (RB, July 1981). With 4K EPROM cards (RB, September 1981), 8K of EPROM can easily be connected to the KIM. Table 1 shows how this memory space is laid out. From address E100 up to and including E382 we find an existing program, namely "Letters on the graphic display", written by Mr D. M. de Boer. From address E000 you will find the key input program dealt with in the previous part, and from address E020 the separately described program "Lines on the graphic display" by Mr R. Koekoek. We will now explain the various controls and commands in more detail using the corresponding listings.

Lijst 1

		1830 ;	
		1840 ;	
		1850 ;	
		1860 ;	ROUTINE VOOR LUIDSPREKERSTURING
		1870 ;	
		1880 ;	
E8FA-	20 9C E3	1890 LUIDS	JSR LSAAN ;INITIALISEER
E8FD-	A5 8E	1900	LDA #GELENG ;HAAL LENGTE VAN GELUID
E8FF-	8D 47 17	1910	STA TIMER2+107 ;
E902-	A6 8F	1920 LUSG	LDA #GEFRE0 ;HAAL FREQUENTIE
E904-	EA	1930	NOP ;
E905-	EA	1940	NOP ;
E906-	EA	1950	NOP ;
E907-	CA	1960 LUSH	DEX ;TEL AF
E908-	D0 FD	1970	BNE LUSH ;
E90A-	EE 00 17	1980	LDC PAD ;TOGGLE LUIDSPREKER
E90C-	A0 47 17	1990	LDA TIMER2+107 ;TIMER KLAAR?
E910-	10 F0	2000	BPL LUSG ;
E912-	4C 96 E3	2010	JMP LUIT ;LUIDSPREKER UIT

Loudspeaker

Listings 1 and 2 show a simple piece of software. First there is a jump to the subroutine LSAAN, where the value \$06 is put in port B. This connects bit 0 of port A to the loudspeaker, because LPH, a pin on the graphic display, becomes active as

discussed earlier. Then a tone with an on/off ratio of 50 % is generated. The duration can be set from about 0.001 to about 0.25 s and the frequency from 500 to about 20000 Hz. Before calling the program you must put the duration at address 008E and the cycle time (1/frequency) at address 008F. The duration is loaded into the timer, which gives a division factor of 1024. X is loaded with the cycle time. In a loop X is decremented by 1 each time until it has become zero. After that PA0 is "toggled", which produces a tone. This continues until the time is up. To switch the loudspeaker off, \$00 is loaded into port B, which makes LPH go "high". Before this can work, the data direction registers of the I/O ports must of course be set correctly. This is done in the routine INITIO, see listing 3. Where there are NOPs now, a JSR instruction to a key input routine will be placed later.

Lijst 2

```

0070 ;
0080 ;
0090 ; ROUTINE SCROLL UP
00A0 ;
00B0 ;
E384- 18 0020 SCRUP CLC ;
E385- 08 0030 CLD ;
E386- A5 A9 0040 LDA *SCROLL ;TEL PEGELHOOGTE OP
E388- 65 A6 0050 ADC *PEGELH ;BIJ DE
E38A- 85 A9 0060 SCRSET STA *SCROLL ;REGELTELLER
E38C- A5 A9 0070 SCROUT LDA *SCROLL ;TEL DIT OP
E38E- 8D 00 17 0080 STA PAD ;BIJ HET
E391- A9 05 0090 LDA #05 ;VERTICALE ADRES
E393- 8D 02 17 1000 STA PBD ;VAN HET DISPLAY
E396- A9 00 1010 LSUIT LDA #00 ;ZET PAD GOED EN
E398- 8D 02 17 1020 STA PBD ;LUIDSPREKER UIT
E39B- 60 1030 RTS ;
1040 ;
1050 ;
1060 ;
1070 ; ROUTINE LUIDSPREKER AAN
1080 ;
1090 ;
E39C- A9 06 1100 LSARN LDA #06 ;MARK LPH ACTIEF
E39E- 00 F8 1110 BNE LSUIT+002 ;

```

Scrolling

Listing 2 shows the routine SCRUP, which moves all text on the screen up one line (scroll up). The data in SCROLL (address 00A9) keeps track of how far the text has already been moved up. The data in REGELH (address 00A8) holds the line height. First the carry and decimal bits are cleared to prevent errors. The line height is added to the line counter and

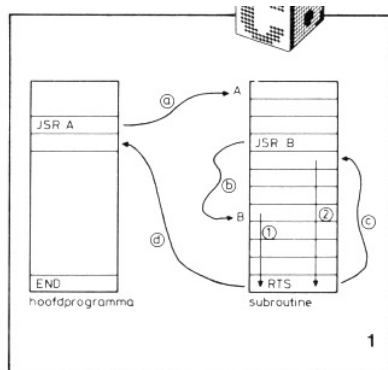
the result is put in SCROLL. This value is loaded into PAD and \$05 into PBD (LPV). As a result the adder ICs add SCROLL to the vertical address of the graphic display. Then PBD is set correctly. This is the same as switching the loudspeaker off, so these two routines could be combined. That saves memory again. With the routine SCRDWN the text on the screen can be moved down one line (scroll down). It works on the same principle as SCRUP, except that the line height is subtracted from the line counter.

Cursor control

The cursor control routine shown in listing 4 is not of the type that puts the cursor in the right place each time the routine is called, but a routine that is run through several hundred times per second. So you have to imagine that some things, such as addresses and registers, were already set correctly on a previous pass. XCOR (address 00A1) holds the current X coordinate and YCOR (address 00A2) the current Y coordinate of the cursor. Suppose the routine had already been called once. In the routine OUT, XCOR and YCOR would then have been copied to LASTX and LASTY respectively; in other words, the last cursor position handled would have been made equal to the position to be handled. If another program has not changed the X and Y coordinates of the cursor (the cursor does not need to be moved), then on the next pass through the routine LASTX equals XCOR and LASTY equals YCOR. The routine OUT is then not executed. The Y coordinate is put in register Y and the X coordinate in register X. It is, however, checked whether the cursor must be inverted; after all, it has to blink. CAAN (address 00AB) records whether the cursor was on or off. At address E3C2 the cursor counter (CCOUNT) is incremented and compared with the value for the blink rate (CSPEED). If they are equal, the cursor is inverted at INVERT. CCOUNT is loaded with \$FB (-5), which keeps track of how many dots must be set; five in this case. CAAN (address 00AB) is set correctly and register Y is incremented to select the right place on the screen; the cursor has to go below the character, after all. Then LUS7 is run five times, writing the dots on the screen, with the result that CCOUNT has become zero.

Now suppose that the X and Y coordinates were changed in another program. Then there is a jump to OUT. Here it is checked whether the value in CAAN is even or odd. If it

is odd, the cursor was on, and there is a jump to INVERT to switch the cursor off at the old position. The cursor at the new position is switched on when the cursor control routine is run a second time. If the old cursor was already off, then only LASTX is made equal to XCOR and LASTY to YCOR. By then making CCOUNT equal to CSPEED, the cursor becomes visible immediately at the new position.



Printing pointers, bytes, nibbles and characters

The routine in listing 5 makes the utmost use of so-called "nested loops". This produces a piece of code that is three times as compact as comparable routines, and that is a goal of every true programmer. Because the structure is rather complex, its operation is explained using the written-out version in listing 6. This is the order of the instructions as the computer encounters them. Because parts of the main program

have to be executed several times, jumps are made to subroutines that are not really subroutines. The stack keeps track of where the next instruction is. Fig. 1 gives an example of a "nested loop". When the computer jumps to a subroutine, the program counter is put on the stack. On return from the routine, the contents of the stack are loaded into the program counter and the computer continues at the indicated place. If, however, a jump is made to a subroutine that is inside program A itself (JSR B) and that also ends with an RTS, the result is that routine B is executed twice. The print routine works in the same way.

One more thing. When a digit has to be printed, it must first be converted to ASCII. The code for "0" is \$30, for "1" it is \$31, and so on up to 9. The ASCII codes for letters do not follow on exactly from those for digits. "A" has the code \$41. To go from 9 to A, as happens when counting in hexadecimal, you have to add \$07 to the code for 9 ($41 - 3A = 7$). This conversion method is also used in this program. When the number \$2B has to be printed, it is first shifted four places to the right: \$2B becomes \$02. This number is less than \$0A, so \$30 is added: $\$02 + \$30 = \$32$ (ASCII code for "2"). The "2" is printed and then \$2B is fetched back into the accumulator from the stack. An AND with \$0F isolates the highest [sic; lowest] four bits: \$2B becomes \$0B. This number is greater than \$0A, so \$37 is added: $\$0B + \$37 = \$42$ (ASCII code for "B"). Finally the "B" is printed.

Table 1

Address	Label	Data	Description
E000	TOETS		; routine (RB, December 1981, p. 38?
E014	ZWA	A9 18	; write black
E016		D0 06	
E018	WIT	A9 10	; write white
E01A		D0 02	
E01C	INV	A9 08	; write inverted
E01E	SCHKL	85 AF	; save in LKLEUR
E020 LIJN			; routine (RB, February 1982, p. 92)
E056		B9 C3 E0	; these three instructions must be
E07A		A5 AF	; changed in the routine "Lines on the
E0C0		4C 69 E0	;graphic display"
E0CB	CLS		; routine "clear screen"
E0E1	PRPNTX		; print pointer
E0EA	PRBYTE		; print byte
E0F3	PRHEX		; print hex byte
E0FE	PRASC		; print ASCII character
E100	LETTER		; routine (RB, June 1979, p. 47)
E2D4		20 58 E3	; these five instructions must be
E2E5		20 58 E3	;changed in the program "Letters on the
E2FE		20 58 E3	;graphic display" by Mr De Boer
E338		20 58 E3	
E34C		65 A8	
E383	SCRUP		; scroll up
E3A0	SCRDOWN		; scroll down
E3A9	INITIO		; initialise input and output
E3B4	CURS		; cursor control
E400	ASCTAB		; ASCII character table (see table 2)
E542			; up to and including E71F must contain \$FF
E720	MANTAB		; ASCII table continued
E780			; up to and including E800 must contain \$FF

Listing 6

```
B5 00      ;GET FIRST BYTE
48         ;SAVE
B5 01      ;GET SECOND BYTE
48         ;SAVE
4A 4A 4A 4A ;TAKE HIGHEST FOUR BITS
C9 0A      ;GREATER THAN 9 ?
18         ;
30 02      ;
69 07      ;IF SO ADD $37
69 30      ;IF NOT, ADD $30
85 A0      ;SAVE IN "LETTER"
20 00 E1   ;SUBROUTINE LETTER
68         ;GET SECOND BYTE
29 0F      ;TAKE LOWEST FOUR BITS
C9 0A      ;GREATER THAN 9 ?
18         ;
30 02      ;
69 07      ;IF SO, ADD $37
69 30      ;IF NOT, ADD $30
85 A0      ;SAVE IN "LETTER"
20 00 E1   ;SUBROUTINE LETTER
68         ;GET FIRST BYTE
48         ;SAVE
4A 4A 4A 4A ;TAKE HIGHEST FOUR BITS
C9 0A      ;GREATER THAN 9 ?
18         ;
30 02      ;
69 07      ;IF SO, ADD $37
69 30      ;IF NOT, ADD $30
85 A0      ;SAVE IN "LETTER"
20 00 E1   ;SUBROUTINE LETTER
68         ;GET FIRST BYTE
29 0F      ;TAKE LOWEST FOUR BITS
C9 0A      ;GREATER THAN 9 ?
18         ;
30 02      ;
69 07      ;IF SO, ADD $37
69 30      ;IF NOT, ADD $30
85 A0      ;SAVE IN "LETTER"
```

20 00 E1 ;SUBROUTINE LETTER

Lijst 3

```

1120 ;
1130 ;
1140 ;
1150 ; ROUTINE SCROLL DOWN
1160 ;
1170 ;
1180 SCREEN SEC ;
1190 CLD ;
1200 LDA #SCROLL ;TREK REGEELHOOGTE AF
1210 SEC #REGELH ;VAN DE TELLER
1220 JMP SCRSCT ;
1230 ;
1240 ;
1250 ;
1260 ; INITIALISEER DE IN- EN UITVOER
1270 ;
1280 ;
1290 INITIO LDA #1F ;
1300 STA PBCO ;
1310 LDA #1FF ;
1320 STA PADD ;
1330 RTS ;

```

Lijst 4

```

1340 ;
1350 ;
1360 ;
1370 ; ROUTINE VOOR CURSORBESTURING
1380 ;
1390 ;
1400 CURS LDX #LASTX ;ZIJN DE HUIDIGE
1410 LDX #LASTY ;ZIJN Y-COORDINAAT
1420 CPX #XCOR ;GELIJK AAN DE
1430 BNE OUT ;ALASTSTE
1440 CPY #YCOR ;ZIJN Y-COORDINAAT?
1450 BNE OUT ;
1460 LDA #COUNT ;MOET DE CURSOR AL
1470 INC #COUNT ;WORDEN
1480 CMP #CSPEED ;GEINVERTEERD?
1490 BNE OUT #01 ;NEE, DAN TERUG
1500 LDA #1FB ;ZIJN JA, INVERTEER
1510 STA #COUNT ;ZET AANTAL PUNTJES
1520 INC #CORN ;ZET VLAG GOED
1530 INX ;VOLGENDE PLAAST
1540 LDA PBD ;MAGHT OP SYNC
1550 BPL LUS7 ;
1560 STX PAD ;AFLOF ADRESSEN
1570 INC PAD ;
1580 INC PAD ;
1590 STY PAD ;
1600 LDA #1FB ;
1610 INC PAD ;
1620 STA PBD ;
1630 INC ;
1640 INC #COUNT ;VOLGENDE PUNT
1650 BNE LUS7 ;
1660 RTS ;
1670 OUT LDA #CORN ;MOET ER WORDEN
1680 LSR A ;GEINVERTEERD?
1690 BCS INVERT ;ZIJN JA, DOE DAT
1700 LDA #XCOR ;ZET DE NIEUWE
1710 STA #LASTX ;CURSOR OP
1720 LDA #YCOR ;ZIJN PLAATS
1730 STA #LASTY ;
1740 LDA #CSPEED ;ZET DIRECT AAN
1750 STA #COUNT ;
1760 RTS ;

```

Lijst 5

```

2440 ;
2450 ;
2460 ;
2470 ; ROUTINE VOOR PRINTEN VAN POINTERS
2480 ; NIBBLES, BYTES EN KARAKTERS
2490 ;
2500 ;
2510 PRPNTX LDA #PPOINTX ;AARAL BYTE
2520 PRA ;BEWAAR
2530 LDA #PPOINTX+101X ;AARAL VOLGENDE BYTE
2540 JSR PRBYTE ;DRUK AF
2550 PLA ;
2560 PRA ;BEWAAR
2570 LSR A ;AARAL HOOGSTE
2580 LSR A ;VIER BITS
2590 LSR A ;
2600 LSR A ;
2610 JSR PRHEX+102 ;
2620 PLA ;AARAL BYTE OP
2630 PRHEX AND #10F ;AARAL LAAGSTE VIER BITS
2640 CMP #10A ;10 9
2650 CLC ;
2660 BMI NEX1 ;
2670 ADC #107 ;ZIJN JA, TEL #17 OP
2680 NEX1 ADC #130 ;ZIJN NEE, TEL #170 OP
2690 PRASC STA #LETTER ;ZET OP SCHERM

```

Lijst 6

```

B5 00 ;AARAL EERSTE BYTE
48 ;BEWAAR
B5 01 ;AARAL TWEDE BYTE
48 ;BEWAAR
4A 4A 4A 4A ;AARAL HOOGSTE VIER BITS
C9 0A ;GROTER DAN 9 ?
18 ;
30 02 ;
69 07 ;ZIJN JA, TEL ER #17 BJT OP
69 30 ;ZIJN NEE, TEL ER #30 BJT OP
B5 00 ;BEWAAR IN "LETTER"
20 00 E1 ;SUBROUTINE LETTER
68 ;AARAL TWEDE BYTE
29 0F ;AARAL LAAGSTE VIER BITS
C9 0A ;GROTER DAN 9 ?
18 ;
30 02 ;
69 07 ;ZIJN JA, TEL ER #17 BJT OP
69 30 ;ZIJN NEE, TEL ER #30 BJT OP
B5 00 ;BEWAAR IN "LETTER"
20 00 E1 ;SUBROUTINE LETTER
68 ;AARAL EERSTE BYTE
4A 4A 4A 4A ;AARAL HOOGSTE VIER BITS
C9 0A ;GROTER DAN 9 ?
18 ;
30 02 ;
69 07 ;ZIJN JA, TEL ER #17 BJT OP
69 30 ;ZIJN NEE, TEL ER #30 BJT OP
B5 00 ;BEWAAR IN "LETTER"
20 00 E1 ;SUBROUTINE LETTER
68 ;AARAL EERSTE BYTE
29 0F ;AARAL LAAGSTE VIER BITS
C9 0A ;GROTER DAN 9 ?
18 ;
30 02 ;
69 07 ;ZIJN JA, TEL ER #17 BJT OP
69 30 ;ZIJN NEE, TEL ER #30 BJT OP
B5 00 ;BEWAAR IN "LETTER"
20 00 E1 ;SUBROUTINE LETTER

```

Captions part 2:

Listing 1 Routine for controlling the loudspeaker.

Listing 2 Routine for the "scroll up" operation.

Table 1 Layout of the memory space of the first EPROM.

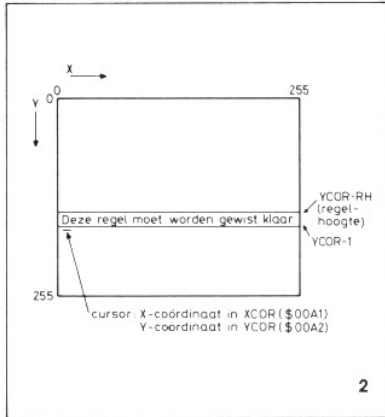
Listing 3 Routine for "scroll down" and the initialisation of input and output.

Listing 4 Routine for controlling the cursor.

Listing 5 Routine for printing pointers, bytes, nibbles and characters.

Listing 6 Written-out version of the print routine.

Fig. 1 Example of a "nested loop" using JSR instructions.



Wait routine

The wait routine shown in listing 7 starts with three NOP instructions, which must not be removed, because a jump to a key input routine will be placed there later. After Y has been saved, a double counting loop follows. The value in X determines the length of the delay. The contents of the X register are therefore lost.

Lijst 7

```

1480 ;
1490 ;
1500 ;
1510 ; ROUTINE VOOR WACHTTIJDEN
1520 ; JRA #E8DC
1530 ;
1540 ;
E8DC- EA 1550 WAIT NOP ;
E8DD- EA 1560 NOP ;
E8DE- EA 1570 NOP ;
E8DF- A2 04 1580 WAIT LDV #04 ;URSTE WACHTTIJD
E8E1- 96 1590 WAIT TYP ;
E8E2- 48 1600 PHA ;BEWAAR Y
E8E3- A0 FF 1610 LUS11 LDV #FF ;255
E8E5- 88 1620 LUS10 DEV ;
E8E6- D0 FD 1630 BNE LUS10 ;
E8E8- CA 1640 DEV ;
E8E9- D0 F8 1650 BNE LUS11 ;
E8EB- 68 1660 PLA ;HAAL Y
E8EC- A8 1670 TAY ;
E8ED- 60 1680 RTS ;

```

Lijst 8

```

2280 ;
2290 ;
2300 ;
2310 ; MARK SCHERM SCHOON
2320 ;
2330 ;
E0CB- A9 1C 2340 CLS LAR #1C ;ZET MISCODE
E0CD- 80 02 17 2350 CLS1 STA PED ;IN POORT B
E0CE- A9 00 2360 LAR #00 ;ZET CURSORLAG
E0D2- 85 A6 2370 STA +CARH ;UIT
E0D4- A0 04 2380 LDV #04 ;LUS VIERMAAL
E0D6- 20 DC E8 2390 LUS JSR WAIT ;WACHT OP TOETS
E0D9- 88 2400 DEV ;
E0DA- D0 FA 2410 BNE LUS ;
E0DC- A9 00 2420 SCRNUL LAR #00 ;HERSTEL SCROLL
E0DE- 4C 8A E3 2430 JMP SCSET ;

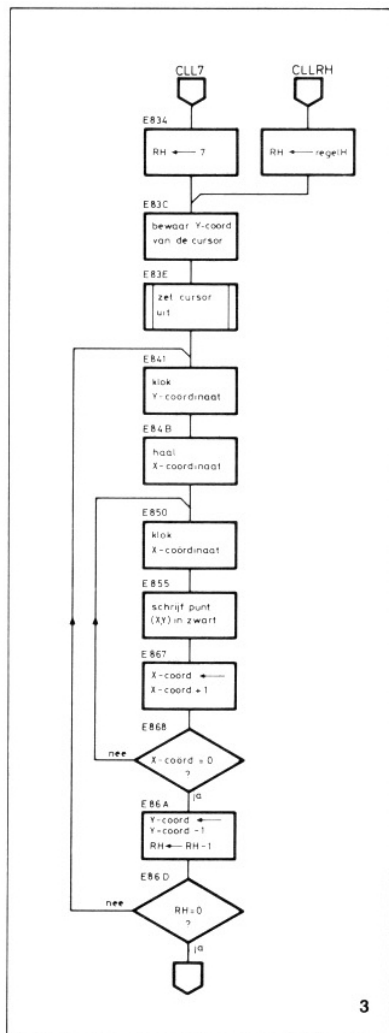
```

Lijst 9

```

0930 ;
0930 ;
1000 ;
1010 ; ROUTINE CLEAR LINE
1020 ;
1030 ;
1040 ;
E834- A2 07 1048 CLL7 LDV #07 ;
E836- D0 02 1050 BNE CLRRA#02 ;0000
E838- A6 A8 1060 CLRRA LDV #FEDEL ;
E83A- A5 A2 1070 LAR #YCOR ;HAAL Y-COÖRDINAAT
E83C- 85 9F 1080 LUSB STA YC ;BEWAAR
E83E- 20 EE E8 1090 JSR RESULT ;
E841- A9 02 1100 LUSC LAR #02 ;SCHRIF OP GO
E843- 80 02 17 1110 STA PED ;
E846- A5 9F 1120 LAR #Y ;
E848- 80 00 17 1130 STA PED ;
E84B- A4 A1 1140 LDV #XCOR ;
E84D- EA 1150 NOP ;
E84E- EA 1160 NOP ;
E84F- EA 1170 NOP ;
E850- A0 02 17 1180 LUSA LAR PED ;WACHT OP SYNC
E853- 10 FB 1190 BPL LUSA ;
E855- A9 01 1200 LAR #01 ;
E857- 80 02 17 1210 STA PED ;
E85A- 8C 00 17 1220 STV PED ;
E85D- A9 03 1230 LAR #03 ;
E85F- 80 02 17 1240 STA PED ;
E862- A9 18 1250 LAR #18 ;
E864- 80 02 17 1260 STA PED ;
E867- C8 1270 INV ;VOLGENDE PUNT
E868- D0 E6 1280 BNE LUSA ;X-COÖRDINAAT = 0 ?
E86A- C6 9F 1290 DEC YC ;VERLAAG POINTER
E86C- CA 1300 DEK ;YH = 0 ?
E86D- D0 02 1310 BNE LUSC ;
E86F- 60 1320 RTS ;

```



Clearing the screen

The routine CLS in listing 8 is more complicated than you might think. First the clear code (\$1C) is loaded into port B, after which there has to be a wait. At first sight this waiting is programmed rather long-windedly. It seems pointless to run the wait routine four times when the value for X could just as well have been made four times as large. The answer is simple. The jump goes to WAITK because a key may be pressed while waiting for the screen to be cleared. During these periods there are therefore four jumps to the key input routine, which is still to be discussed. With some monitors you can only give a new command from the keyboard once the previous command has been executed. That is not the case here, because the key input routine stores a pressed key until a running program has finished.

When the screen has been cleared, the cursor must also be off. This is done by setting CAAN to zero. The routine ends with a jump to SCRSET, because after the screen has been cleared the scrolling must be set correctly. The adder ICs in the graphic display are filled with \$00.

Clear a line

The routine in listing 9 is a good deal more interesting and is explained with the help of fig. 2. The cursor shown has the coordinates (X,Y), in other words (XCOR, YCOR). To clear the line concerned, $[RH*256-XCOR]$ dots must be made black. The value in YCOR is put in register YC. This is the Y coordinate of the dot to be written black (see also fig. 3). The cursor is switched off and the Y coordinate is sent to the display. Then the X coordinate is incremented a number of times and the dots concerned are written black. When X has become zero after passing \$FF, Y and YC are decremented by 1. This repeats until YC has become zero. [Note: the code decrements YC and X, and stops when RH (in X) reaches zero.]

Clear to the end of the screen

If you have understood the above, you will immediately see how the routine in listing 10 works. RH must become $SCROLL-YCOR-1$. Now you will be thinking: why should RH not be equal to $-YCOR-1$? Suppose you are working with the monitor and the screen has scrolled. The bottom of the screen then has SCROLL as its Y coordinate, because if everything has been moved up by, say, ten dots, the dots that disappear at the top reappear at the bottom. Because in the routine CLEOS everything down to the bottom of

the screen must be cleared, the Y coordinate must be subtracted from SCROLL (the number of dots by which the screen has been moved up vertically). Then 1 more must be subtracted, because everything from the cursor onwards must be cleared. At the same time RH is put in X and SCROLL in YC, which corresponds to moving the cursor to the bottom of the screen. This was done so that the routine LUSB, which is jumped to, does not need to be changed in any essential way. There, everything above the cursor is cleared up to RH lines above the cursor. By making RH equal to the Y coordinate of the bottom of the screen minus the Y coordinate of the cursor minus 1, and making the Y coordinate of the cursor equal to that of the bottom of the screen, a routine "clear to the end of the screen" is created that corresponds to the routine "clear a line".

Cursor off

The only thing the routine in listing 11 does is put the X and Y coordinates of the cursor in X and Y respectively and jump to the routine OUT. This is part of the cursor control routine in listing 4. The X register is left intact.

ASCII table

Completely separate from the software discussed is the ASCII table. This table of ASCII characters for the graphic display was broadcast by Hobbyscoop (a NOS programme) together with the program "Letters on the graphic display", but never published. That has now been done. The first column of table 2 gives the ASCII code for the character. The second column shows the shape of the character. The third column contains the code for the way in which the character must be printed. This has been discussed before, so we will not go into it further. These values are placed in the EPROM, starting at address E720. The fourth column gives the data needed to fill the character matrix. One byte represents one column of dots. If it is \$01, the top dot of the column is switched on; the bottom dot if it is \$80. It then depends on the value in the third column whether the whole must be rotated or mirrored. The values printed in column four are placed in the EPROM, starting at address E400.

Lijst 10				Lijst 11			
	1330	:			1690	:	
	1340	:			1700	:	
	1350	:			1710	:	
	1360	:	ROUTINE CLEAR TILL END OF SCREEN		1720	:	ROUTINE CURSOR UIT
	1370	:			1730	:	
	1380	:			1740	:	
E870- D8	1390	CLEOS	CLD ;	E8EE- 8A	1750	CRSUIT	TXA ;
E871- 18	1400		CLC ;	E8EF- 48	1760		PHA ;BEWAAR X
E872- A5 A9	1410		LDA *SCROLL ;	E8F0- A6 AD	1770		LDX *LASTX ;AAAL COORDINATEN
E874- A6	1420		TAY ;	E8F2- A4 AE	1780		LDY *LASTY ;
E875- E5 A2	1430		SEC *YCOR ;SCROLL - YCOR - 1	E8F4- 20 EE E3	1790		JSR OUT ;JDER UIT
E877- AA	1440		TAY ;	E8F7- 68	1800		PLA ;
E878- 88	1450		DEY ;VOLGENDE REGEL	E8F8- AA	1810		TAX ;X TERUG
E879- 98	1460		TYA ;	E8F9- 60	1820		RTS ;
E87A- 4C 3C E8	1470		JMP LUSB ;MAAK REGEL SCHOON				

Table 2

ASCII value Character		ASCII-table 2 ASCII table 1	
hex		hex	hex
20	space	82	00 00
21	!	01	BE
22	"	A2	0E 00
23	#	A3	28 FE 28
24	\$	05	48 54 FE 54 24
25	%	05	C6 26 10 C8 C6
26	&	05	6C 92 AC 40 A0
27	'	01	0E
28	(	03	38 44 82
29	)	03	82 44 38
2A	*	A3	44 28 FE
2B	+	A3	10 10 7C
2C	,	22	80 60
2D	-	82	10 10
2E	.	01	80
2F	/	05	C0 20 10 08 06
30	0	05	7C A2 92 8A 7C
31	1	03	8C FE 80
32	2	04	C4 A2 92 8C
33	3	04	44 92 92 6C
34	4	04	1E 10 10 FC
35	5	04	9E 92 92 62
36	6	04	7C 92 92 64
37	7	04	C2 22 12 0E
38	8	82	6C 92
39	9	04	4C 92 92 7C
3A	:	01	28
3B	;	02	80 68
3C	<	04	10 28 44 82
3D	=	82	28 28
3E	>	04	82 44 28 10
3F	?	05	04 02 B2 12 0C
40	@	05	7C 82 BA B2 9C
41	A	82	FC 12
42	B	04	FE 92 92 6C
43	C	04	7C 82 82 44
44	D	04	FE 82 82 7C
45	E	04	FE 92 92 82
46	F	04	FE 12 12 02
47	G	04	7C 82 A2 64
48	H	82	FE 10

49	I	A2	82 FE
4A	J	04	60 80 80 7E
4B	K	05	FE 10 28 44 82
4C	L	04	FE 80 80 80
4D	M	A3	FE 04 08
4E	N	05	FE 04 08 10 FE
4F	O	82	7C 82
50	P	04	FE 12 12 0C
51	Q	05	7C 82 A2 42 BC
52	R	04	FE 12 32 CC
53	S	04	4C 92 92 64
54	T	A3	02 02 FE
55	U	82	7E 80
56	V	A3	3E 40 80
57	W	A3	7E 80 70
58	X	A3	C6 28 10
59	Y	A3	06 08 F0
5A	Z	05	C2 A2 92 8A 86
5B	[	04	FE 82 82 82
5C	\	05	06 08 10 20 C0
5D	]	04	82 82 82 FE
5E	↑	A3	08 04 FE
5F	—	61	F0
60	`	03	00 01 02
61	a	04	40 A8 A8 F0
62	b	04	FE 88 88 70
63	c	03	70 88 88
64	d	04	70 88 88 FE
65	e	04	70 A8 A8 10
66	f	03	08 FC 0A
67	g	24	1C A2 A2 7E
68	h	04	FE 08 08 F0
69	i	01	FA
6A	j	03	40 80 7A
6B	k	04	FE 20 50 88
6C	l	03	82 FE 80
6D	m	05	F8 08 F0 08 F0
6E	n	04	F8 08 08 F0
6F	o	82	70 88
70	p	24	FE 22 22 1C
71	q	24	1C 22 22 FE
72	r	04	F8 10 08 08
73	s	04	90 A8 A8 48
74	t	03	08 7E 88
75	u	82	78 80

76	v	A2	78 80
77	w	A3	78 80 60
78	x	A3	88 50 20
79	y	24	1E A0 A0 7E
7A	z	04	88 C8 A8 98
7B	{	03	10 6C 82
7C		01	FF
7D	}	03	82 6C 10
7E		43	80 70 08
7F	␣	04	AA 54 AA 54

Captions part 3:

Fig. 2 How the routine "clear a line" works.

Table 2 ASCII table.

Listing 7 Routine for waiting times.

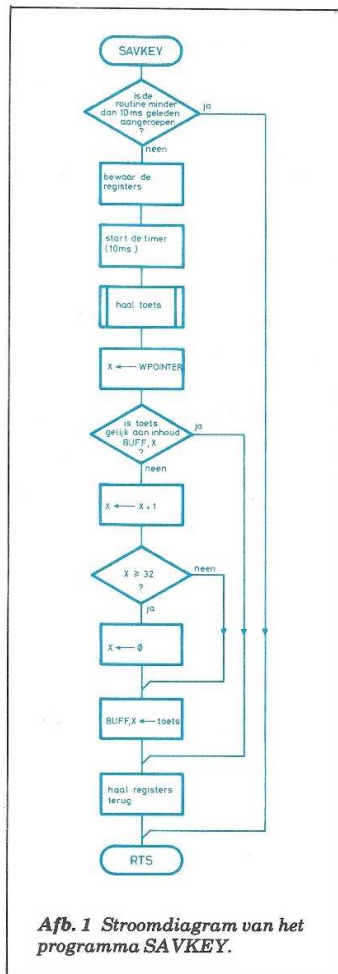
Listing 8 Routine for clearing the screen.

Listing 9 Routine for clearing a line.

Listing 10 Routine for clearing to the end of the screen.

Listing 11 Routine for switching the cursor off.

Fig. 3 Flowchart of the routine "clear a line".



In the previous parts we described a number of routines that were needed for the monitor. The treatment of one special routine was left out, however. This routine, the key input routine, is now discussed in full.

As you may have noticed, a number of routines (CURSOR, WAIT and WIS LIJN [clear line]) contained three NOPs. We said that these would later be replaced by a JSR instruction. We could now replace these NOPs with JSR KEYIN, which starts at address \$E000 (discussed in part 1). That, however, would be taking the easy way out. A jump to that key input routine in programs such as CURSOR, WAIT and WIS LIJN would seem very illogical. The key input routine we have made consists of three parts:

1. Get a key and save it.
2. Create a repeat function.
3. Execute saved keys.

Practice has shown that many computers have no repeat function and that this is a great loss; think of the "CRLF" key or the "+" key of the KIM. A second drawback, found when trying out various systems, is that when a key is pressed while a program is running (for example DRAW LINE or CLEAR SCREEN), it is not interpreted by the computer. Also, when several keys are pressed in

succession, at most only the first is executed. The key input routine described here has an automatic repeat function. Furthermore, up to a total of sixteen keys can be stored while a program is running, and these are then executed one after the other once it has finished. At first sight sixteen keys seems a lot, but if you consider that drawing a figure on the screen takes a relatively long time, it is certainly desirable to be able to "remember" up to sixteen keys.

Key input routine

Fig. 1 shows the flowchart of this program. Use is made of the timer. This was done to make the routine run somewhat faster. In the routine WAIT, for example, this routine is called five times, fairly quickly one after the other. To avoid running the routine again for nothing, there is a timer that measures the time between this call of the routine and the previous one. If it was called less than 10 ms ago, there is no point in looking at the keyboard. If it was more than 10 ms ago, the registers are put on the stack and the timer is restarted. Then the keyboard is read. This is done twice to suppress contact bounce. Now that we have read the key, it still has to be saved. This can only be done in one way, namely with a FIFO buffer. FIFO is the abbreviation of an English term, First In First Out,

which means that the value that went into the buffer first must also come out first. This is in contrast to the stack of a microprocessor, which is a LIFO buffer (Last In First Out). The FIFO needs:

1. A piece of memory.
2. A "pointer" that indicates where the last value read out is (RPOINT).
3. A "pointer" that indicates where the last value read in is (WPOINT).

In the routine SAVKEY we do not yet have anything to do with these two pointers. We simply put the key in the next empty place, provided it is a new key. For suppose the key was still pressed. The key would then be put in the buffer again. The result would be that within 0.1 s the whole FIFO buffer would be filled with one key value. But we have another problem. Suppose the key is pressed twice in succession. The previous key was then equal to the new one, so the latter would not be put in the buffer. This problem can be solved in two ways:

1. The key input routine waits until the key has been released again. A first drawback, however, is that the waiting holds up other programs. A second drawback is that the repeat function is then no longer easily possible.
2. It is indicated that a new key has been pressed. (In Basic you meet something similar with the CR-LF key, which indicates that a new line is being entered.) A drawback here is that after every key, another key has to be used to indicate that a new key is about to be pressed (think of the "+" key on the KIM, which selects the next address).

Even so, we chose method 2, although its drawbacks seem greater. This drawback has, however, been removed by creating a new key, namely the "none" key. Every time a key is pressed, the "none" key automatically follows. In the key input routine \$FF is put in A if no key was pressed. By treating the "none" key as a separate key, everything stays "real time" and a repeat function is possible. Instead of 16 bytes, the FIFO buffer now does become 32 bytes long, because after every key \$FF must also be stored to indicate that the key was released. The whole program is now easy to understand (listing 1).

REPEAT or SAVKEY

The second program we discuss in this part is a good deal harder to understand than the first. Here two programs are interwoven. A flag determines which of the two programs is run:

1. Create the repeat function.
2. Execute SAVKEY.

This flag, called TELLER [counter], is at address \$0081. If its value is \$00, SAVKEY is executed, which means that the key that was put in the buffer first is fetched. If TELLER is not zero, REPEAT is executed. The principle used here is roughly the same as that of the Mini-assembler for the 6502 (RB, March and April 1981), in which a key got several functions depending on how long it was pressed, which was kept track of by a counter. In this way it is determined here how fast the repeat runs. If TELLER is zero, there is no repeat. In the following explanation of the program we will assume that for the moment.

Lijst 1

```

1150 ; ROUTINE BEWAAR TOETS
1160 ;
1170 ;
E800- 20 07 17 1180 SAVKEY BIT TIMER ;ZIJN DE 10 MS AL OM?
E803- 10 2E 1190 BPL RET ;
E805- 48 1200 PHA ;BEWAAR
E806- 8A 1210 TVA ;REGISTERS
E807- 48 1220 PHA ;OP DE
E808- 98 1230 TVA ;STACK
E809- 48 1240 PHA ;
E80A- A9 0A 1250 LDA #0A ;START TIMER OPTIEUW
E80C- 00 07 17 1260 STA TIMER ;10-24 MS>
E80F- 20 20 02 1270 DEBOH JSR GETKEY ;INDIRECTE SPRONG NAAR
1280 ;TOETSOPHAALROUTINE VIA RAM
E812- 85 84 1290 STA #8KEY ;OM DENDER TE
E814- 20 20 02 1300 JSR GETKEY ;ONDERDRIJVEN
E817- C5 84 1310 CMP #8KEY ;LOSSELATEN?
E819- D0 F4 1320 BNE DEBOH ;NEEN, OPNIEUW
E81B- A6 83 1330 LDA #APOINT ;IS HET AANGEGEVEN
E81D- D0 00 02 1340 CMP BUFF,X ;KARAKTER UIT DE
E820- F0 0C 1350 BRQ OUT ;BUFFER GELIJK?
E822- E8 1360 INH ;NEEN, VOLGENDE PLAATS
E823- E8 20 1370 CPA #820 ;BUFFER VOL?
E825- 90 02 1380 BCC BUFNU ;NEEN, ZET TOETS OF /GEEN TOETS?
E827- A2 00 1390 LDA #00 ;RESET BUFFER
E829- 90 00 02 1400 BUFNU STA BUFF,X ;IN DE BUFFER
E82C- 86 83 1410 STX #APOINT ;BEWAAR POINTER
E82E- 68 1420 OUT PLA ;HAAL
E82F- A8 1430 TVA ;REGISTERS
E830- A8 1440 PLA ;TERUG
E831- A8 1450 TVA ;VIN DE
E832- 68 1460 PLA ;STACK
E833- 68 1470 RET RTS ;KEER TERUG
1480 ;
1490 ;
1500 ;LS
1504 ;
0220- 4C 00 00 1505 GETKEY JMP TOETS ;INDIRECTE ADRESSERING VAN
1506 ; TOETSOPHAALROUTINE VIA RAM

```

Lijst 2

```

1530 ; ROUTINE VOOR REPEAT-FUNCTIE
1540 ; OF OPELAAN VAN EEN TOETS
1550 ;
1560 ;
1570 ;
E870- A0 00 1580 RESKEY LDV #00 ;GEEN REPEAT
E87F- 84 81 1590 RKEY STX #TELLER ;ZET TELLER
E881- A6 82 1600 SAVKEY LDV #APOINT ;ZIJN DE POINTERS
E883- E4 83 1610 CPA #APOINT ;VAN ELKANDER GELIJK?
E885- F0 12 1620 BRQ REPT ;ZO JA, REPEAT
E887- E8 24 1630 INK ;NEEN, VOLGENDE BUFFERPLAATS
E888- E0 20 1640 CPA #E20 ;BUFFER VOL?
E88A- 90 02 1650 BCC NIBUF ;
E88C- A2 00 1660 LDA #00 ;JRL RESET BUFFER
E88E- 86 82 1670 NIBUF STX #APOINT ;ZET POINTER
E890- 80 00 02 1680 LDA BUFF,X ;HAAL TOETS UIT BUFFER
E893- 20 00 08 1690 JSR SAVKEY ;NAAR BEWAARROUTINE
E896- 4C 82 08 1700 JMP EXIT ;TERUG
E899- A6 81 1710 REPT LDV #TELLER ;HAAL TELLER
E89B- C0 1720 DEN ;HARS TELLER = 1?
E89C- F0 1C 1730 BRQ RTAN ;ZO JA, TERUG
E89E- E8 1740 INK ;HARS TELLER = 0?
E89F- F0 02 1750 BRQ BPREV ;ZO JA, VIA GEFDE
E8A1- C6 81 1760 DEC #TELLER ;NEEN, REPEAT
E8A3- 20 20 02 1770 BPREV JSR GETKEY ;HAAL TOETS
E8A6- 85 84 1780 STA #KEY ;BEWAAR DE TOETS
E8A8- 20 23 02 1790 JSR SROUT ;HAAL HART UIT
E8AB- 20 20 02 1800 JSR GETKEY ;VOORRON DENERDEN
E8AE- C5 84 1810 CMP #8KEY ;
E8B0- D0 F1 1820 BNE BPREV ;
E8B2- C5 80 1830 EXIT CPA #KEY ;HARS HET DETELENDE TOETS?
E8B4- F0 08 1840 BRQ SAVKEY ;ZO JA, OPNIEUW
E8B6- 85 80 1850 STA #KEY ;ZET TOETS NAAR
E8B8- 38 C3 1860 BMT RESKEY ;HARS HET AFDO
E8BA- 4C 26 02 1870 RTAN JMP TRKEY ;NEEN, LIND

```

Captions part 4:

Fig. 1 Flowchart of the program SAVKEY.

Fig. 2 Flowchart of the program RESKEY.

Listing 1 Program SAVKEY.

Listing 2 Program RESKEY.