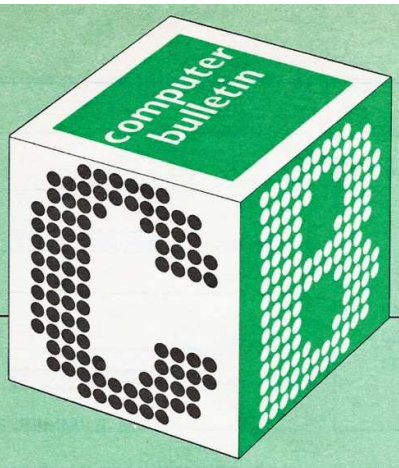


computer bulletin

een supplement van RB, gewijd aan microprocessoren en aanverwante onderwerpen

NIEUWS

Onze rubriek microgebeuren vindt u weer op pagina's 40, 45 en 46

SOFTWARE

Een praktische toepassing van de microcomputer vindt u in het artikel 'Morse decodering met KIM'. De geseinde tekst verschijnt op het display van de KIM. Het is evenwel ook mogelijk met wat wijzigingen dit programma te laten lopen op andere 6502 machines. (blz. 36)

GRAFISCH DISPLAY

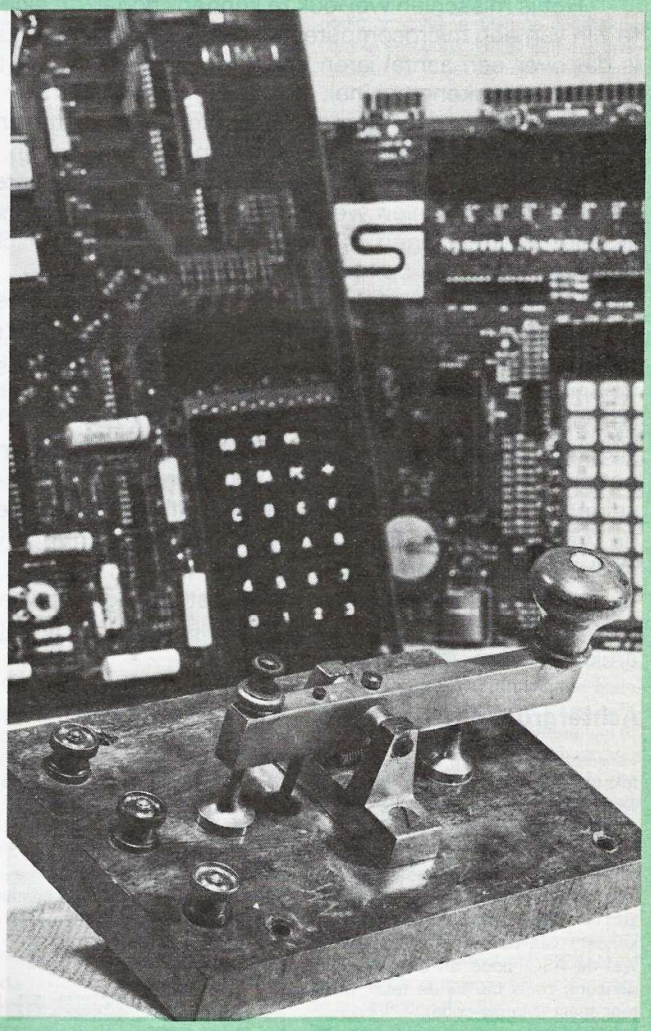
Door ruimtegebrek moest vorige maand de bij het programma behorende symbol table komen te vervallen. Deze symbol table, plus een correctie op een drukfout in lijst 1, vindt u op blz. 39

TEST

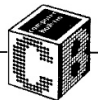
Deze maand komt het Heathkit H8 systeem aan de beurt. Zoals we van Heathkit gewend zijn géén compleet apparaat, maar een zéér goed gedocumenteerde bouwdoos. (blz. 41)

CURSUS

Het laatste deel van de cursus Basic kunt u vinden op blz. 47



Morsedecodering is voor de microcomputer een koud kunstje...



MORSE-DECODING WITH THE KIM

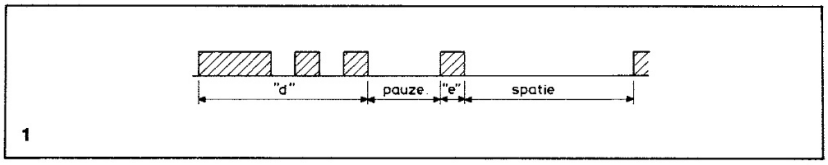
M.B. Immerzeel

It may be quite nice to be able to say that you are the owner of a microcomputer (if things continue as they are now, in a few years this will be just as commonplace as owning a pocket calculator is today), but even nicer is to also own programs and to be able to spend a number of useful and above all pleasant hours with the device. The following program, written for the KIM, may contribute to this. It makes it possible to display received Morse signals as 'readable writing' on the KIM display. Although with a seven-segment display by no means all letters and punctuation marks can be formed, and therefore certain symbols had to be found for these characters, after some practice it is quite possible to read messages sent in Morse code from the display. The program can be used not only for displaying received messages, but also for practising sending and for checking the sending technique. From what is shown on the display one can determine what mistakes are being made while sending. A variant of this program, namely displaying Morse writing with a teletype, may be expected in a possible following article.

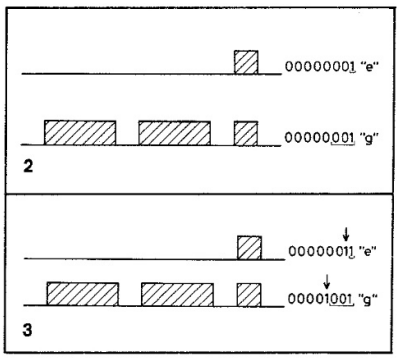
Background

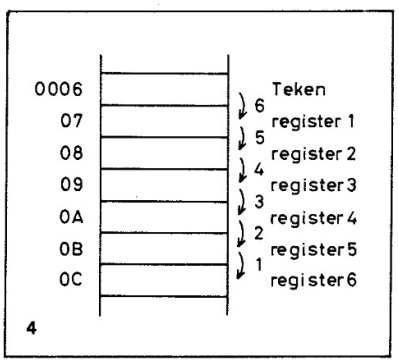
For transferring written texts by mechanical-electrical means (telex) or via modern communication methods (teletext or viewdata, for example), it is necessary to use a coding system in which the writing and reading characters are each represented by a separate code character. The best-known of these coding systems are the ASCII code and the five-unit code as used in telex (Murray code). The advantage of these systems is that all code characters are of equal length and each code character is divided into an equal number of long 'units' (five in telex), which are also called 'elements'. Each element can then be 'current flowing' or 'currentless', comparable to the '1' and '0' of the binary system. The specific characteristics of this system, namely that each code character always consists of a fixed number of elements of equal length and that these can only be '0' or '1', make it relatively easy for the machine to recognize the correct

written characters from the supplied code characters.
The situation is quite different with the Morse code system. It might be said that this system is easier for a human being to analyse with his hearing than the codes mentioned above, but for the machine it is difficult to digest the fact that the code characters are irregular both with regard to the number of elements (from one dot for the ‘e’ to six dots for the error character) and with regard to the length of the elements. The dashes in these code characters are, after all, three times as long as the dots. It is not only the dots and dashes that determine the character. Between the elements of the character there is a silence which we will further call ‘rest’, and whose length should be equal to that of a dot. Between the code characters a longer pause must be maintained. This pause must last as long as a dash. A code character for a space is generally not transmitted. For this, a longer pause is simply used which must have at least a length of five dots (seven dots are prescribed now, formerly five dots were used).
In summary, the word ‘de’ in Morse characters looks as in fig. 1. The ‘d’ is a dash with two dots. The rests are one dot long and the dash is three dots. The ‘e’ is only one dot and is separated from the ‘d’ by a pause with a length of three dots. The next word will only begin after a pause with a duration of seven dots. Another problem requiring a solution is the sending speed, which differs for every telegraphist. It is therefore necessary first to determine the length of a dot. This is only possible after both a dot and a dash have been received. This program will therefore not display the first character (or in some cases the first few characters), or will display it incorrectly. This cannot be a serious problem. Every transmission is preceded by a call which is repeated several times. If a sending text has not been generated by a machine, in almost all cases it will not look as prescribed. We will have to take into account that a dash is not always three dots long. In this program, the length of two dots has been taken as the limit, i.e. every element longer than two dots is recognized as a dash. This does mean that a sender can no longer be ‘copied’ if he sends dots that are more than twice as long as previously sent dots.
Furthermore, the program must take irregularities in sending speed into account. Increasing the sending speed causes no problems here. However, reducing the sending speed must, in connection with the above, be such that the length of the next dot must in any case be less than twice the length of the preceding one.
To continue following these changes in sending speed, it is necessary always to remember the length of the last received dot. It can, however, happen that an uncontrolled key movement by a less experienced sender causes a very short contact, which is then interpreted as a very short dot. If this ‘dot’ lasts less than half the duration of the following dots, all following code elements will be regarded as dashes, so that the operation of the program will then become completely confused. This disturbance can be prevented by averaging the duration of every received dot with that of the preceding one. In this way the outliers are ‘smoothed out’ and irregularities have less influence on correct operation.



It has already been stated that the sending speed can be increased without limit, but cannot be reduced without limit. Nevertheless, a stepwise speed change can be handled reasonably well. Taking the aforementioned ‘averaging’ into account, a step from, for example, 23 to 16 words per minute can therefore be allowed without difficulty. Regular reductions in speed can therefore also cause no problems.





Lijst 1

Teken	Morse-code	Inh. 'teken' binair	Inh. 'teken' hex.	Display	Display-code hex.
A	•■	00000110	06		F7
B	■•••	00010111	17		FC
C	■•■•	000101001	15		B9
D	■••	00001011	0B		DE
E	•	00000011	03		F9
F	••■•	00011101	1D		F1
G	■■•	00001001	09		EF
H	••••	00011111	1F		F4
I	••	00000111	07		90
J	•■■■	00011000	18		8E
K	■•■	00001010	0A		F0
L	•■••	00011011	1B		B8
M	■■	00000100	04		B7
N	■•	00000101	05		D4
O	■■■	00001000	08		DC
P	•■■•	00011001	19		F3
Q	■■•■	00010100	12		E7
R	•■•	00001101	0D		D0
S	•••	00001111	0F		ED
T	■	00000010	02		F8
U	••■	00001110	0E		9C
V	•••■	00011110	1E		BE
W	•■■	00001100	0C		FE
X	■••■	00010110	16		F6
Y	■•■■	00010010	14		EE
Z	■■••	00010011	13		DB
1	•■■■■	00110000	30		86
2	••■■■	00111000	38		DB
3	•••■■	00111100	3C		CF
4	••••■	00111110	3E		E6
5	•••••	00111111	3F		ED
6	■••••	00101111	2F		FD
7	■■•••	00100111	27		87
8	■■■••	00100011	23		FF
9	■■■■•	00100001	21		EF
0	■■■■■	00100000	20		BF
?	••■■■■•	01110011	73 (33)		A7
,	■■••■■	01001100	4C (2B)		8C
/	■••■•	00101101	2D		D2
.	•■•■•■	01101010	6A (1A)		88
:	■■■•••	01000111	47 (26)		89
—	■••••■	01011110	5E (3D)		C0
=	■•••■	00101110	2E		c—
spatie	•■•••■	00110110	36		80
begin	■•■•■	00101010	2A		D0
eind	•■•■•	00110101	35		C4
sluiten	••••■■	01111010	7A (3A)		80
apostroph	•■■■■•	01100001	61 (11)		82
(	■•■■•■	01010010	52 (31)		8F
wacht	•■•••	00110111	37		C9
	■•■•■•	01010101	55 (34)		8D

Lijst 2

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0			T	E	M	N	A	I	O	G	K	D	W	R	U	S
1	.	(A_p)	Q	Z	Y	C	X	B	J	P	(.)	L		F	V	H
2	0	9		8			(:)	7			Be	(,)		/	=	6
3	1	(()		(?)	(;)	E_i	S_p	W_a	2		(S_t)		3	(–)	4	5
4								:					'			
5			(			;									–	
6		A_p									.					
7				?							S_t					

The display

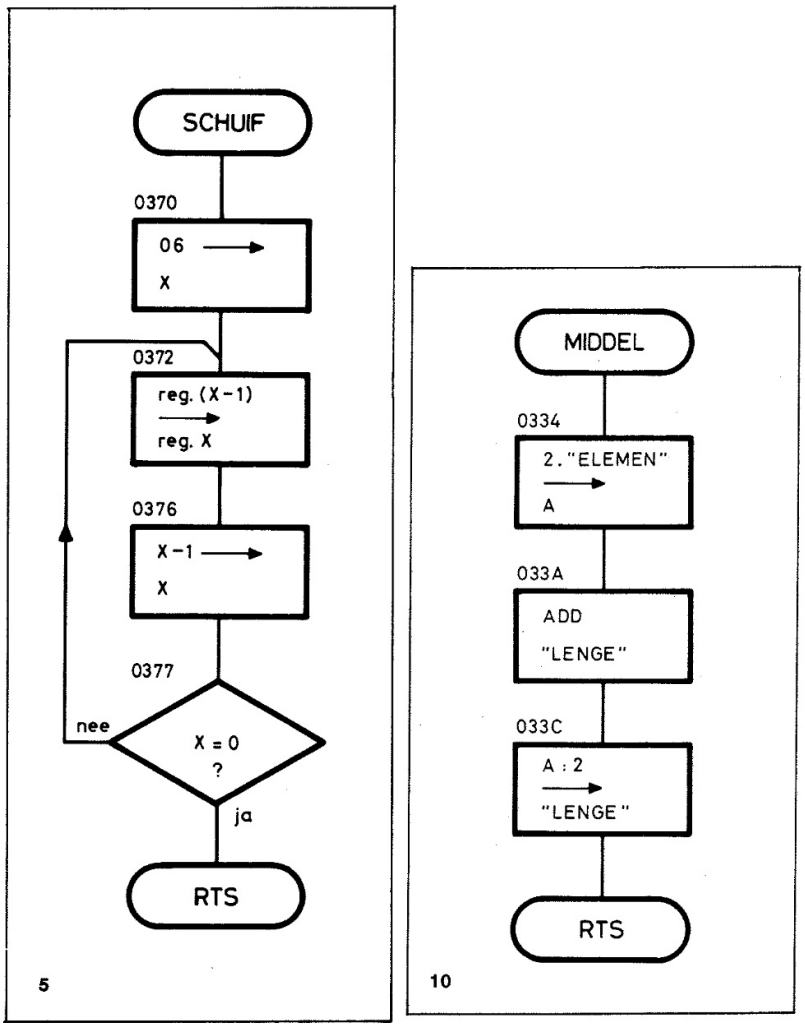
This program was made to display the Morse characters as reading characters on the KIM display. The difficulty, however, is that it is only a seven-segment display and therefore cannot display certain letters and punctuation marks. For these letters and punctuation marks (k; m; n; v; w; x; ?; comma; period and semicolon), a number of symbols have therefore been devised which resemble the original letters and characters as much as possible. It has been found that after a relatively short period of practice the texts, despite the strange symbols, can also be read smoothly at high sending speeds.

The display shows the texts as happens with a lighted sign. One difference, however, is that the text always shifts by one position. The characters appear one after another on the rightmost display and, after reception and translation of the next Morse character, always shift one position to the left. A space is displayed by leaving the corresponding display blank.

In list 1, among other things, the punctuation marks used, with their corresponding Morse characters and display symbols, are given. For translating the Morse characters, a dot is read as '1' and a dash as '0'. For the KIM too it is necessary to switch to code characters that all contain the same number of elements. Here the number of elements is: 8, namely equal to the number of bits per memory location.

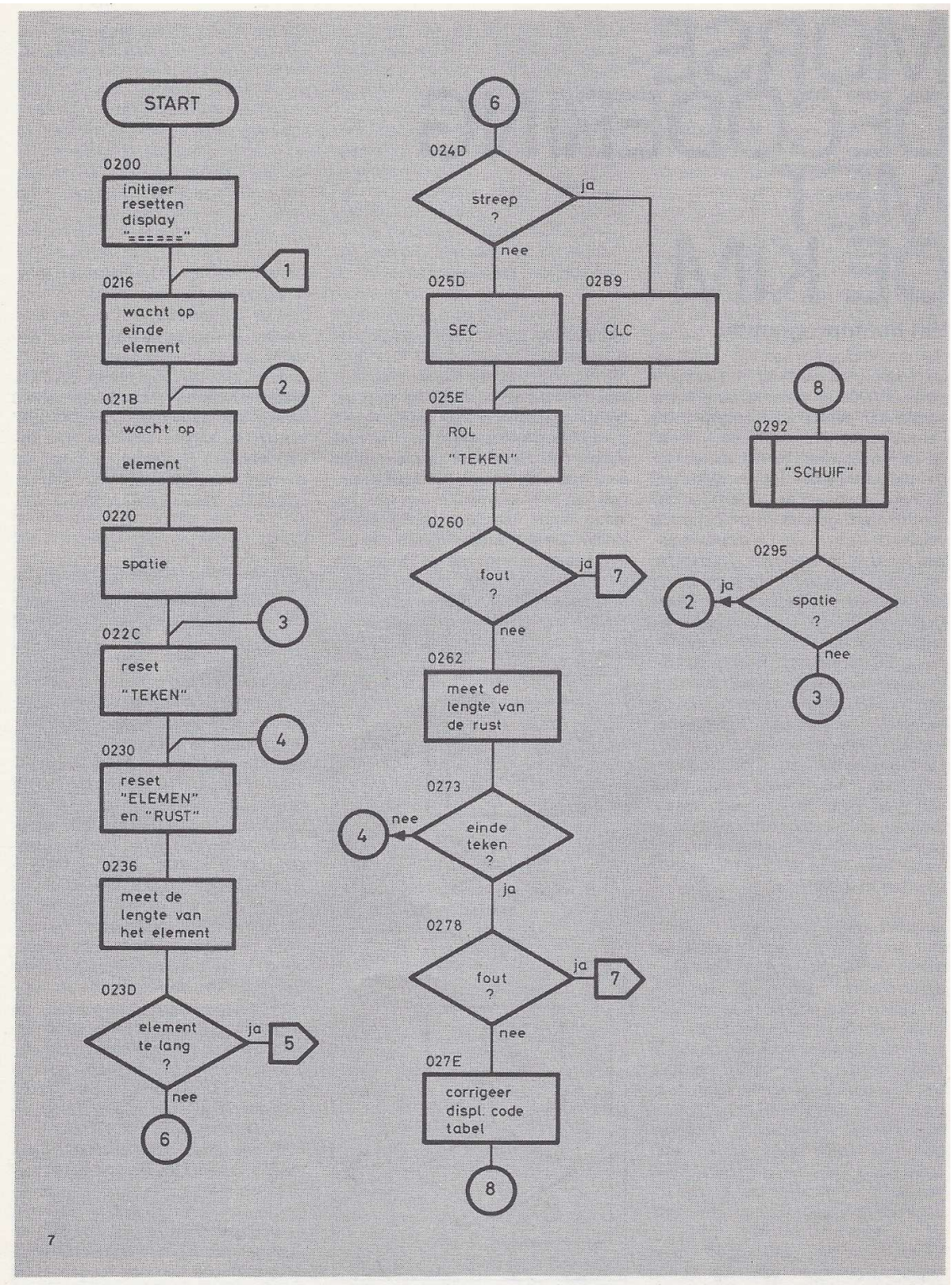
Because all empty memory locations are provided with a '0', errors would arise if the Morse characters were simply translated into ones and zeros. There would then, for example, no longer be any distinction between an 'e' and a 'g', because the dashes ('0') and the empty memory locations (also '0') could no longer be distinguished. Fig. 2 shows how an 'e' and a 'g' would be read by the KIM. By marking the characters beforehand with a '1', this error is prevented. Fig. 3 shows how the 'e' and the 'g' are translated by the KIM. The markings are indicated by an arrow.

In list 1, under the column 'CHARACTER', the code characters into which the KIM translates the Morse character are shown. Each Morse character is therefore translated into a binary number, whose hexadecimal value is shown in the column 'character HEX'. At address 0006 we find the memory location 'TKEN'. The elements of the code character are shifted into this one after another. When the character is complete, it is transferred to the first of the six registers used by the monitor program. Memory lcations 0007 through 000C are reserved for this (fig. 4).



The subroutine 'SCHUIF' is then activated, whereby, starting with register 5, the contents are shifted one position and the lighted-sign effect is thus achieved. The memory location 'TEKEN' is now available again for the next character (fig. 5).
It was not possible to use the 'SCANDS' subroutine of the KIM monitor program. The first reason is that the program is also used when measuring the length of the elements and a different delay is required than that used in the CONVD subroutine. The second reason is that now many more than sixteen characters have to be displayed, so that more than four bits are needed per code character — and also for the display code.
The flow diagram for the 'DISPLAY' subroutine is outlined in fig. 6 and broadly corresponds to that of 'SCANDS'.
The 'character HEX contents' column of list 1 gives as its lowest value hex 02 for the 'T' and has hex 7A for 'close' as its highest value (see also list 2). This would mean that almost 128 memory locations would be required for the display-code table, with more than 64 unused locations (there are 51 different code characters).
The size of the table can be reduced by 'moving' certain characters. It follows from list 2 that for this purpose the characters on row 7 must be reduced by hex 40, those on row 6

by hex 50 and those on rows 5 and 4 by hex 21. In list 1 the new values are placed in parentheses. The display-code table is in memory locations 02C0-02FF.



The main program

The flow diagram of the main program is shown in fig. 7. The start address is 0200 and the sending key is connected in parallel with one of the keyboard contacts (e.g. A17 and A18 of the application connector). After starting the program by pressing the 'go' key, the input and output buffers are first initialized and certain auxiliary registers are reset. The display registers are loaded with the code character for '=', after which the program waits until the 'go' key is released. After release, the program waits for the first code element.

For these last two program steps the display subroutine is used so that the display lights up. Immediately after pressing the sending key, i.e. at the start of an element, the rightmost display goes out because 'TEKEN' is loaded with the code character for 'space' (3A), which is placed in register 1 by means of subroutine 'schuif'.

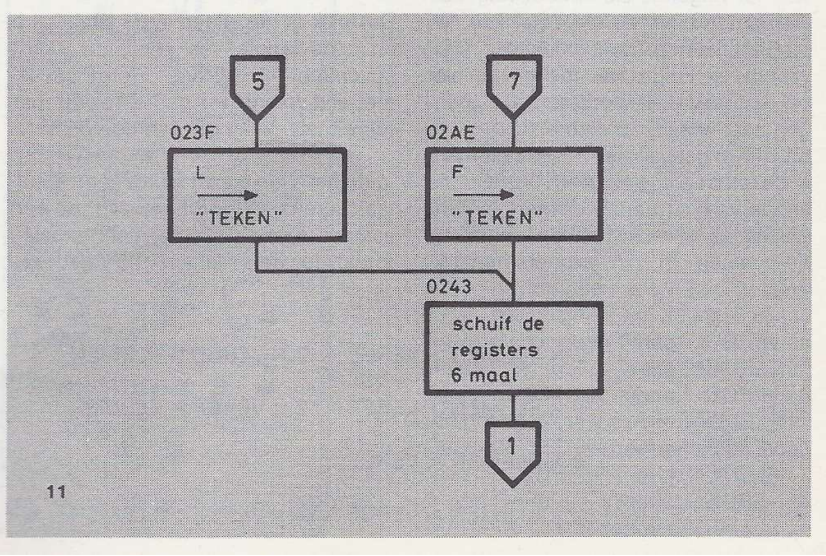
The program step 'reset character' loads 'TEKEN' with hex 01 for the marking. The 'DISPLAY' subroutine is also used for measuring the length of the element; it is run through a number of times and during each cycle the counter 'ELEMEN' is increased (fig. 8). The display program must therefore have a certain length and memory location 031E is loaded with hex FF for the correct delay.

At high sending speeds (30 words per minute and faster), it may be better to load this address with hex 7F. The delay is thereby reduced so that a larger number of measurements takes place in a given time.If the sending key is held down too long continuously, an error indication follows (the eighth bit of 'ELEMEN' becomes 1).At the end of the element, that is, at the beginning of the following rest, it is checked whether the element was a dot or a dash. This is obviously not possible after receiving the first element. Only after two elements of different lengths have been received can this measurement take place. The subroutine 'VERGELIJK' (see later) ensures that the length of a dot is calculated. Memory location 'LENGE' (0000) is then filled with twice the length of a dot. The contents of 'ELEMEN' are compared with this. If the element is longer (dash), CLC follows; otherwise SEC. The program step 'ROL teken' now shifts respectively a 0 or a 1 into the first bit of 'TEKEN', while all other bits shift to the left (fig. 9). Furthermore, if the element proves to be a dot, subroutine 'MIDDEL' calculates the average value of the new length of the dot with the previous one (fig. 10).

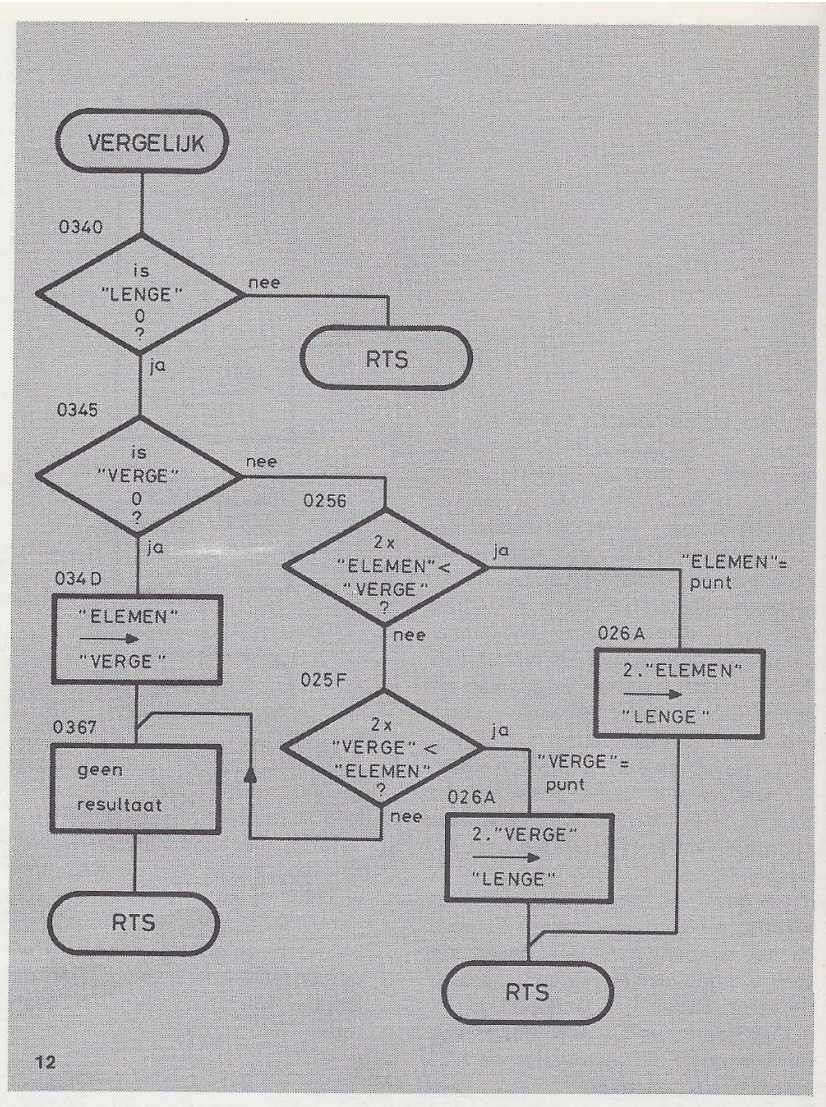
The greatest number of elements from which a Morse character can consist is six. If we add the marking character, a maximum of seven bits of the eight bits of memory location 'TEKEN' are therefore needed. If the marking character nevertheless reaches the eighth bit, this is an error and an error indication follows. This error will occur with telegraphists who do not leave enough space between the Morse characters and thus 'stick' everything together.

The length of the rest is determined in a corresponding manner to the length of the element. The flow diagram is therefore also the same as that in fig. 8. For element and 'ELEMEN', however, 'RUST' must now be read. The counter is now memory location 'RUST' at address 0005.

In theory, the length of the rest is equal to that of the dot. It is better, however, not to rely on this and always average the length of a rest with the preceding one, just as with the dot. This method is therefore also applied in this program, although for the first character, when the length of the rest is not yet known, the length of the dot is taken as the standard. In practice this proves to work well and guarantees that the first correct character appears on the display in the shortest possible time.
Memory location 'LENGE' at address 0003 is then filled with twice the length of a rest. The contents of 'RUST' are compared with this and it will then become clear whether it is really a rest or a pause.
In the case of a rest (end of character is: no), after resetting 'ELEMEN' and 'RUST' we proceed to measuring the new element which has now begun. In the case of a pause, the code character is complete (end of character).
Just as with the element, this pause can be held for a very long time, causing the 'RUST' counter to overflow. This is by no means considered an error and in this case the KIM automatically gives a space.The next program step determines whether the telegraphist has sent an error indication (six dots). If so, the display gives 'FFFFFF'. Otherwise it is checked whether the contents of 'TEKEN' must be changed to limit the size of the display-code table (list 2), after which subroutine 'SCHUIF' shifts the contents of the display registers and puts the contents of 'TEKEN' into the first display register.As far as the length of the pause is concerned, there are now two more possibilities. The pause lasts approximately three times as long as the rest. In this case, after resetting 'TEKEN', 'ELEMEN' and 'RUST', we proceed to measuring the first element of the next code character, which has already begun.
The pause is longer (seven times as long as the rest or much longer). This means the end of a word. The program now optionally waits for the first element of the next code character and will then, after first giving a space, reset as under a and measure the length of that element.If everything is correct, the pause between two letters of a word lasts approximately as long as three rests. This pause can then be found by comparing with four times the length of a rest. Most telegraphists, however, make this pause too long (for fear of 'sticking'?). The result could be that the KIM gives a space after every letter. It is therefore better to compare with six times the rest. The subroutine 'LANG' at address 0388 is intended for this, and can be called by placing hex 8B at address 0298. For those who want to practise sending with the KIM, it is also possible to compare with four rests ('short') or five rests ('medium') by placing respectively hex 7C or 80 at address 0298.
In the foregoing, there was mention three times of an error indication: once for holding the sending key too long, then for too many elements in 'TEKEN', and finally for sending the code character for 'error'. In the first case the display shows 'LLLLLL' and in the other cases 'FFFFFF'. For this purpose the relevant code character is loaded into 'TEKEN' and, by running through the subroutine 'SCHUIF' six times, is placed in all six display registers. The program now returns to 'wait for end of element' (fig. 11).


5
023F
L
"TEKEN"
7
02AE
F
"TEKEN"
0243
schuif de
registers
6 maal
1

11

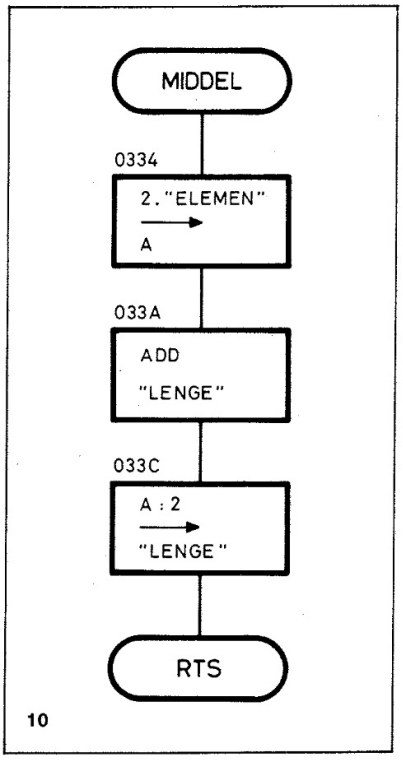

VERGELIJK
0340
is
"LENGE"
0
?
nee
RTS
ja
0345
is
"VERGE"
0
?
nee
ja
034D
"ELEMEN"
"VERGE"
0256
2x
"ELEMEN"<
"VERGE"
?
ja
"ELEMEN"=
punt
nee
026A
2."ELEMEN"
"LENGE"
025F
2x
"VERGE" <
"ELEMEN"
?
ja
"VERGE"=
punt
nee
026A
2."VERGE"
"LENGE"
0367
geen
resultaat
RTS
RTS

12

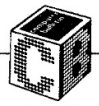
The subroutine ‘vergelijk

The compare subroutine is used exclusively, when receiving the first code elements, to determine which element is a dot. Once the length of a dot is known and ‘LENGE’ has been loaded with a value that is twice 20 [the source OCR reads ‘20’ here] times as large, this subroutine no longer serves any purpose. Right at the start of the subroutine it is therefore determined whether ‘LENGE’ is loaded or not (fig. 12). After receiving the first element, ‘LENGE’ and ‘VERGE’ are still empty and the contents of ‘ELEMEN’ are loaded into ‘VERGE’ (0001). Obviously it is not yet known whether this is a dot or a dash. The contents of ‘VERGE’ are further used to compare the length of the next element or next elements with it. The program step ‘geen resultaat’ establishes that the length of the dot is not yet known, after which it jumps back to ‘wait for element’. After receiving the second element, the situation is therefore that the length of the first element is recorded in ‘VERGE’ and the length of the second in ‘ELEMEN’. There are now four possibilities. It is clear that only in situations b and c is the length of a dot found and ‘LENGE’ is filled with a value corresponding to twice the length of a dot. In cases a and d no result is reached and a wait is made for a following element. This will repeat until situation b or c occurs.

a.	VERGE: dot ELEMEN: dot	2 x ELEMEN > VERGE	2x VERGE > ELEMEN
b.	VERGE: dot ELEMEN: dash	2 x ELEMEN > VERGE	2x VERGE < ELEMEN
c.	VERGE: streep ELEMEN: dash	2 x ELEMEN < VERGE	
d.	VERGE: dash ELEMEN: dash	2 x ELEMEN > VERGE	2x VERGE > ELEMEN



10



```
0200 20 8C 1E START  JSR  INIT1      ┐
0203 A9 00           LDA  #$00      │
0205 85 00           STA  LENGE     │ Initialiseren
0207 85 01           STA  VERGE     │
0209 85 03           STA  LENGR     │
020B 85 04           STA  VERGR     ┘
020D A2 06      RESET LDX  #$06     ┐
020F A9 2E           LDA  #$2E      │
0211 95 06      RESET1 STA TEKEN,X  │ DISPLAY '---'
0213 CA              DEX            │
0214 D0 FB           BNE  RESET1    ┘
0216 20 00 03 MORSEA JSR  DISPL     ┐ Wacht op einde
0219 D0 FB           BNE  MORSEA    ┘ element
021B 20 00 03 MORSE1 JSR  DISPL     ┐ Wacht op element
021E F0 FB           BEQ  MORSE1    ┘
0220 A9 3A           LDA  #$3A      ┐
0222 85 06           STA  TEKEN     │ Spatie
0224 20 70 03        JSR  SCHUIF    ┘
0227 EA EA EA        NOP's
022A EA EA           NOP's
022C A2 01      MORSE2 LDX #$01     ┐
022E 86 06           STX  TEKEN     │
0230 A2 00      MORSEB LDX #$00     │ Reset
0232 86 02           STX  ELEMEN    │
0234 86 05           STX  RUST      ┘
0236 20 00 03 MORSE3 JSR  DISPL     ┐
0239 F0 12           BEQ  VERG1     │ Meet de lengte
023B E6 02           INC  ELEMEN    │ van het element
023D 10 F7           BPL  MORSE3    ┘
023F A9 1B           LDA  #$1B      ┐ Element te lang
0241 85 06           STA  TEKEN     ┘
0243 A0 06      MORS11 LDY #$06     ┐
0245 20 70 03 MORS12 JSR  SCHUIF    │ Schuif de
0248 88              DEY            │ registers
0249 D0 FA           BNE  MORS12    │ 6 maal
024B F0 C9           BPL  MORSEA    ┘
024D A2 00      VERG1  LDX #$00     ┐
024F 20 40 03        JSR  VERGEL    │ Bepaal, streep of
0252 F0 C7           BEQ  MORSE1    │ punt
0254 C5 02           CMP  ELEMEN    │
0256 30 61           BMI  MORSE7    ┘ Streep in ELEMEN
0258 A2 02           LDX  #$02      ┐ Punt in ELEMEN
025A 20 34 03        JSR  MIDDEL    │
025D 38              SEC            ┘ Punt is 1
025E 26 06      MORSE8 ROL TEKEN    ┐
0260 30 50           BMI  MORS10    │
0262 20 00 03 MORSE4 JSR  DISPL     │
0265 D0 37           BNE  VERG2     │
0267 E6 05           INC  RUST      │ Meet de lengte
0269 10 F7           BPL  MORSE4    │ van de rust
026B A5 03           LDA  LENGR     │
026D D0 04           BNE  MORSE9    ┘
026F A5 00      MORSE5 LDA LENGE    ┐ Inhoud LENGE in
0271 85 03           STA  LENGR     ┘ LENGR
0273 C5 05      MORSE9 CMP RUST     ┐ Bepaal rust of pauze
0275 10 30           BPL  MORS3A    ┘ Rust
0277 38              SEC            ┐ Pauze (einde teken)
0278 A5 06           LDA  TEKEN     ┘
027A C9 70           CMP  #$70      ┐ Foutmelding?
027C 10 30           BPL  VERM5     ┘
027E C9 60           CMP  #$60      ┐
0280 10 08           BPL  VERM2     │
0282 C9 40           CMP  #$40      │
0284 30 0C           BMI  VERM4     │
0286 E9 21           SBC  #$21      │ Corrigeer de display-
0288 10 06           BPL  VERM3     │ code tabel
028A E9 50      VERM2  SBC #$50     │
028C 10 02           BPL  VERM3     │
028E E9 3F      VERM1  SBC #$3F     │
0290 85 06      VERM3  STA TEKEN    ┘
0292 20 70 03 VERM4  JSR  SCHUIF    ┐ Schuif de registers
0295 A5 03           LDA  LENGR     ┘
0297 20 8B 03        JSR  LANG      ┐ Pauze of spatie
029A 30 20           BMI  JUMP      ┘
029C 10 8E           BPL  MORSE2    ┐ Spatie
029E A2 03      VERG2  LDX #$03     ┘ of pauze
02A0 20 40 03        JSR  VERGEL    ┐
02A3 F0 CA           BEQ  MORSE5    │ Is lengr gevuld?
02A5 D0 CC           BNE  MORSE9    ┘
02A7 A2 05      MORS3A LDX #$05     ┐
02A9 20 34 03        JSR  MIDDEL    │ Rust
02AC D0 82           BNE  MORSEB    ┘
02AE C9 7F      VERM5  CMP #$7F     ┐ Indien geen fout-
02B0 D0 DC           BNE  VERM1     ┘ melding, corrigeer
02B2 A9 D1      MORS10 LDA #$D1     ┐
02B4 85 06           STA  TEKEN     ┘ Foutmelding
02B6 4C 43 02        JMP  MORS11    ┐
02B9 18         MORSE7 CLC          ┘ Streep in elemen
02BA 90 A2           BCC  MORSE8    ┐
02BC 4C 1B 02 JUMP   JMP  MORSE1    ┘ Tussensprong voor 029A
02BF -- 
02C0 -- -- -- F8 TBL                ┐
02C3 F9 B7 D4                       │
02C6 F7 90 DC                       │
02C9 EF F0 DE                       │
02CC FE D0 9C                       │
02CF ED -- 82                       │
02D2 E7 DB EE                       │
02D5 B9 F6 FC                       │
02D8 8E F3 88                       │
02DB B8 -- F1                       │ Displaycodetabel,
02DE BE F4 BF                       │ '--' wil zeggen: inhoud
02E1 EF -- FF                       │ is niet van belang
02E4 -- -- 89                       │
02E7 87 -- -- --                    │
02EA D0 8C -- --                    │
02ED D2 C8 FD                       │
02F0 86 8F -- --                    │
02F3 A7 8D C4                       │
02F6 80 C9 DB                       │
02F9 -- 80 -- --                    │
02FC CF C0 E6                       │
02FF ED                             ┘
0300 A9 7F      DISPL  LDA #$7F     ┐ PAD op uitgangen
0302 8D 41 17          STA PADD.    ┘
0305 A2 09             LDX #$09     ┐ Eerste display
0307 A0 06             LDY #$06     ┘ 6 display's
0309 B9 06 00 DISPLA   LDA TEKEN,Y  ┐ register naar accu
030C 84 FC     DISPL1  STY TEMP     ┘ Save Y
030E A8                TAY          ┐ Haal de display-
030F B9 C0 02          LDA TBL,Y    ┘ code uit de tabel
0312 A0 00             LDY #$00     ┐ Display uit
0314 8C 40 17          STY PAD      ┘
0317 8E 42 17          STX PBD      ┐ Kies display
031A 8D 40 17          STA PAD      ┘ Display aan
031D A0 FF             LDY #$FF     ┐ Laat het display
031F 88        DISPLB  DEY          │ een ogenblik aan
0320 D0 FD             BNE DISPLB   ┘
0322 E8                INX          ┐ Stel X in op het
0323 E8                INX          ┘ volgende display
0324 A4 FC             LDY TEMP     ┐ Stel Y in op het
0326 88                DEY          ┘ volgende register
0327 D0 E0             BNE DISPLA   ┐ Laatste display?
0329 8E 42 17          STX PBD      ┘ Display uit
032C A9 00             LDA #$00     ┐ PAD naar ingang
032E 8D 41 17          STA PADD     ┘
0331 4C FE 1E          JMP AK       ┐ Naar 'Any Key?'
0334 B5 00     MIDDEL  LDA LENGE,X  ┐
0336 18                CLC          │ Vermenigvuldig met 2
0337 0A                ASL          ┘
0338 CA                DEX          ┐ Stel vorige waarde in
0339 CA                DEX          ┘
033A 75 00             ADC LENGE,X  ┐ Tel de vorige waarde op
033C 4A                LSR          ┘ en deel door 2
033D 95 00             STA LENGE,X  ┐ Zet het resultaat weg
033F 60                RTS          ┘ en terug
0340 B5 00     VERGEL  LDA LENGE,X  ┐ Is LENGE of LENGR
0342 F0 01             BEQ VERGE1   ┘ gelijk aan 0?
0344 60                RTS          ┐ nee
0345 E8        VERGE1  INX          ┐
0346 B5 00             LDA LENGE,X  │ Is VERGE of VERGR
0348 F0 03             BEQ VERGE2   ┘ gelijk aan 0?
034A 4C 56 03          JMP VERGE3   ┐ Nee
034D E8        VERGE2  INX          ┐
034E B5 00             LDA LENGE,X  │
0350 CA                DEX          │ Inhoud van ELEMEN
0351 95 00             STA LENGE,X  │ naar VERGE
0353 4C 67 03          JMP VERGE4   ┘
0356 E8        VERGE3  INX          ┐
0357 B5 00             LDA LENGE,X  ┘ ELEMEN in accu
0359 0A                ASL          ┐ x2
035A CA                DEX          ┘
035B D5 00             CMP LENGE,X  ┐ 2. ELEMEN-VERGE
035D 30 0B             BMI VERGE5   ┘ Resultaat negatief?
035F B5 00             LDA LENGE, X ┐ VERGE in accu
0361 0A                ASL          ┘ x2
0362 E8                INX          ┐
0363 D5 00             CMP LENGE,X  ┘ 2.VERGE-ELEMEN
0365 30 03             BMI VERGE5   ┐ Resultaat negatief?
0367 A9 00     VERGE4  LDA #$00     ┘ Geen resultaat
0369 60                RTS          ┐
036A CA        VERGE5  DEX          │
036B CA                DEX          │
036C 95 00             STA LENGE, X │
036E 60                RTS          ┘
036F -- 
0370 A2 06     SCHUIF  LDX #$06     ┐ 6 registers te schuiven
0372 B5 05     SCHUI1  LDA TEKEN-1,X┐ Verplaatst een
0374 95 06             STA TEKEN,X  ┘ register
0376 CA                DEX          ┐
0377 D0 F9             BNE SCHUI1   ┘ Laatste?
0379 60                RTS          ┐ Terug
037A -- -- 
037C 0A        KORT    ASL          ┐ 2 x LENGR (4 x rust)
037D C5 05             CMP RUST     ┘ Vergel. 4 x rust met pauze
037F 60                RTS          ┐
0380 4A        MIDDEN  LSR          ┘ LENGR:2 (1 x rust)
0381 85 0D             STA SAVE     ┐ 1 x rust in SAVE
0383 0A                ASL          ┘ 2 x rust in accu
0384 0A                ASL          ┐ 4 x rust in accu
0385 18                CLC          ┘
0386 65 0D             ADC SAVE     ┐ 5 x rust in accu
0388 C5 05             CMP RUST     ┘ Verg. 5 x rust met pauze
038A 60                RTS          ┐
038B 85 0D     LANG    STA SAVE     ┘ 2 x rust in SAVE
038D 0A                ASL          ┐ 4 x rust in accu
038E 18                CLC          ┘
038F 65 0D             ADC SAVE     ┐ 6 x rust in accu
0391 C5 05             CMP RUST     ┘ Verg. 6 x rust met pauze
0393 60                RTS
```

Gebruikte geheugenplaatsen

0000	LENGE	0007	displayregister 1
0001	VERGE	0008	displayregister 2
0002	ELEMEN	0009	displayregister 3
0003	LENGR	000A	displayregister 4
0004	VERGR	000B	displayregister 5
0005	RUST	000C	displayregister 6
0006	TEKEN	000D	save
		00FC	temp

Discussion of the program

The complete program is given in list 3. The attentive observer will be able to discover a number of minor imperfections in it. The only consequence of these imperfections is that a few more memory locations are used than was strictly necessary. I therefore did not see sufficient reason to rewrite the entire program for this, with the risk of introducing errors which are not always equally easy to trace again.
In brief, some important points follow:

— Start at address 0200.

— Sending key between pins A17 and A18 of the application connector.

— After receiving two different code elements (a dot and a dash), the characters become visible on the display.

— With a very slow sender the display shows ‘LLLLLL’.

— With a ‘sticker’ or an error indication ‘FFFFF’.

— An increase in sending speed is accepted without any problem.1

— A stepwise reduction of less than 1/3 of the sending speed is also accepted. At high sending speeds, hex 7F at address 031E.

— For sending practice, hex 7C at address 0298.

— Interference pulses (e.g. when receiving radio signals) can disturb the program. All elements are then registered as dashes. In that case restart the program.

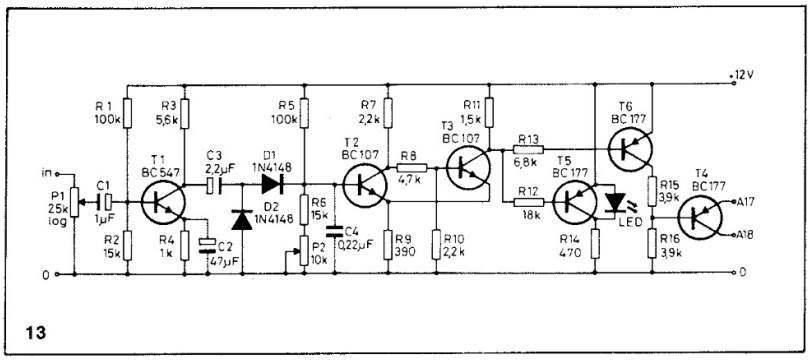
Auxiliary circuit

A circuit by which signals from a receiver can be converted into pulses for the KIM is sketched in fig. 13. The circuit is sensitive but has too low an input impedance to be connected directly to the output of a detector. For the DC voltage, the 12 V supplied by the KIM power supply can be used. The input sensitivity is adjusted with potentiometer P1. The signal tapped from the low-frequency section of the receiver (possibly from the loudspeaker connection) is amplified by T1 and rectified by diodes D1 and D2. Transistors T2 and T3 are connected as a Schmitt trigger. In the rest state T3 is conducting and therefore T5 and T6 as well. The LED is short-circuited by T5 and is therefore off. Because T6 is conducting, T4 is cut off (the KIM experiences this as '0'). The transistor T4 is connected to A17 and A18 of the application connector (note the correct polarity, A17 is emitter).

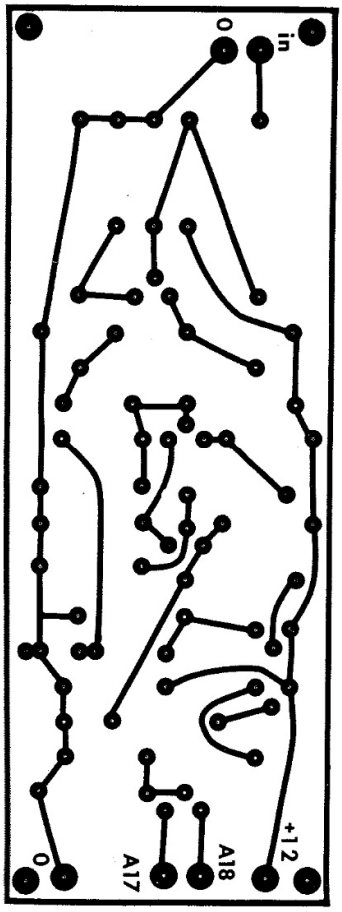
If a sufficiently large signal arrives at the input, T2 conducts. T6 cuts off, causing T4 to open and A17 and A18 to be practically short-circuited. The KIM experiences this as '1'. T5 also cuts off, so that the LED connected in parallel with it lights up.

To adjust P2, make sure there is no signal at the input. Potentiometer P2 is now adjusted so that the LED just does not light. Then connect the signal to the input and turn up P1 until the LED makes the Morse characters visible by lighting.

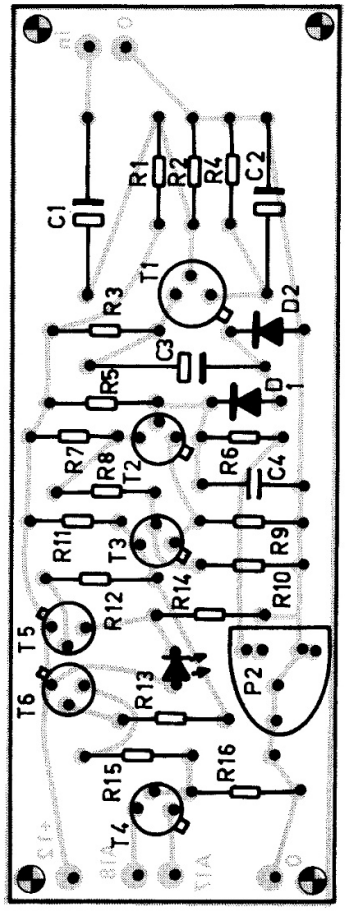
In fig. 14 the PCB is drawn and in fig. 15 the component layout.


+12V
R1 100k
R3 5,6k
R5 100k
R7 2,2k
R11 1,5k
T6 BC177
T3 BC107
R13 6,8k
T1 BC547
C3 2,2µF
D1 1N4148
T2 BC107
R8 4,7k
T5 BC177
R15 3,9k
T4 BC177
in
P1 25k log
C1 1µF
D2 1N4148
R6 15k
R12 18k
LED
A17
A18
R2 15k
R4 1k
C2 47µF
P2 10k
C4 0,22µF
R9 390
R10 2,2k
R14 470
R16 3,9k
0
0

13


0
in
0
A17
A18
+12

14


C1
R1
R2
R4
C2
T1
R3
D2
C3
R5
D1
R7
R8
T2
R6
C4
R11
T3
R9
R14
R10
R12
T5
T6
P2
R13
R15
R16
T4