



LETTERS

ON THE GRAPHIC TV DISPLAY

D. M. de Boer

In the previous series of articles on the home-built graphic TV display, a lot of attention was paid to the hardware of this display. Two important routines were also discussed, namely switching on or off one of the 65536 dots and clearing the picture. But... a display is not a display if no text can be shown on it. That is why this article presents what is perhaps the most important piece of software written for this display: 'drawing' letters, with unprecedented flexibility.

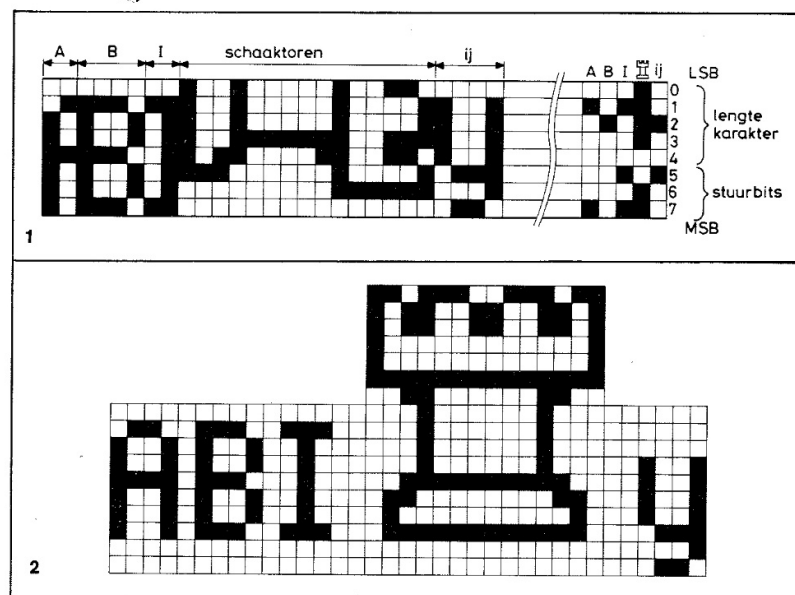
Hardware or software?

Letters on the graphic display can be produced in several ways. First the 'classic' method, a character generator with a special piece of RAM in which the characters are stored in ASCII form. The advantage of this method is that the display is very fast, and that letters can easily be read back from the screen. The drawback of the 'hardware method' is that components have to be bought and that the user is bound hand and foot to the properties of the circuit.

The software solution gives unprecedented flexibility as regards letter shape, letter height, letter width, letter spacing, letter position, etc. Moreover, the solution is cheaper (the whole 'circuit' fits in two 2708 EPROMs). The drawback of software, slowness, turns out not to be too bad. With the 1 MHz 6502 about 125 characters per second can be drawn. The 6800 is a little slower; the 8080 manages about 70 characters per second. This assumes that 48 dots have to be drawn per character (8 x 6 matrix). In this case the letter is drawn white and the background black, or vice versa. If we leave out drawing the background, the whole thing becomes somewhat faster still. One drawback remains: reading a character back from the screen is not simple. This problem can be avoided by storing the ASCII codes of the letters on the screen in the computer's memory.

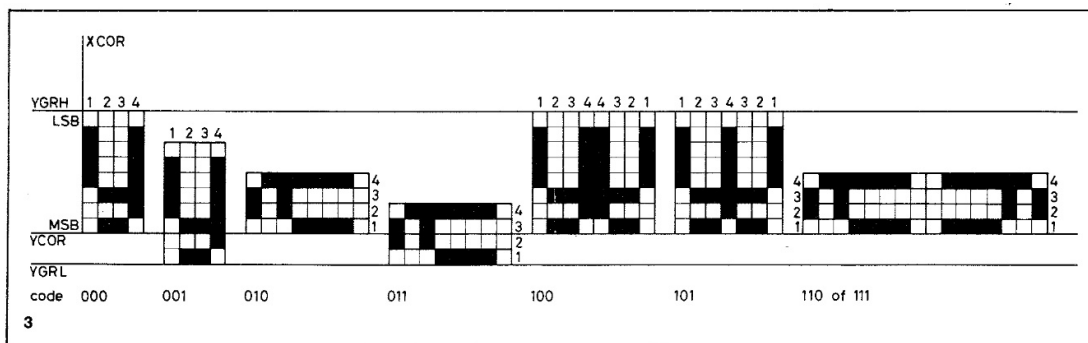
Often it is not necessary at all to read letters back, and the copy is then of course superfluous. The software solution, however, offers a number of very attractive further possibilities. A kind of (very flexible) programmable character generator has, as it were, come into being automatically. The character set can be anywhere in RAM (or EPROM). The characters needed for a chess program can be written along with the chess

All these enormous possibilities made us choose software character generation. Of course it remains possible to use a 'normal' character generator alongside this software and add it to the display hardware. In this way speed and flexibility are combined.



For character generation we need a program that draws characters on the screen in a prescribed shape. It uses a table in which the letter shapes are stored. This table can be regarded as a software character set. The way the letters are stored of course depends on the program used. In the program we developed, we chose maximum freedom as regards character width. The dot grid in which the character must be placed is 8 dots high and can be 1 to 31 dots wide. For 'normal' letters the width will vary from 1 to 5 dots. If a letter has a vertical axis of symmetry (for example A), only half of the letter needs to be in the character generator; the program provides the other half by mirroring. It is also possible to store a character tilted in the character set. Here we are thinking mainly of chess pieces. A tilted character is always 8 dots wide and can be 1 to 31 dots high. If we work with symmetrical figures (vertical axis of symmetry), the figure becomes 16 dots wide and still between 1 and 31 dots high. In this way chess pieces can be stored in great detail. Only the knight is not symmetrical and has to be split into two

parts. In the same way little rockets, moon landers, little figures, etc. can be put into the character generator as characters. All this is clarified in fig. 1. Here you see part of the character table. Bits that are '0' have been left open; bits that are '1' have been filled in. Five different situations are drawn one after the other. Normally the characters are of course in alphabetical order; this order was chosen as an example. Fig. 2 shows how the various characters appear on the screen. Apart from the shape of the letters, the program of course also needs to know where the character can be found in the character set, and how many bytes the character takes up. That is why a table has to be added to the character set from which this information can be obtained. In this table it is indicated for each character how long the character is, and whether the character must be mirrored, rotated and/or shifted. We will go further into how the table works later in this article. Because all the characters are packed tightly together, space is used very efficiently. In a normal character generator all letters are at least 5 cells wide because of the width of the letter 'M'. The four letters above would have taken 20 bytes; now they take only 16 bytes. The chess rook would not have been possible.



Setting options

As we have seen, characters in the character generator can be mirrored, rotated and shifted (see figs. 1 and 2). Because these settings differ per character, they must also be specified for each character. There are, however, also settings that apply to all characters in a given situation. We are thinking here of enlargement factors for height and width, the place where the character is to be put, etc. In the program developed by RB the following settings are possible:

XCOR This memory location gives the horizontal address of the character (0-255).

YCOR This memory location gives the vertical address of the character (0-255).

BREED [width] This memory location gives a factor by which the character width is multiplied. So for BREED=3 the letters become 3 times as wide (0-255).

HOOG [height] The same as BREED, but for the height.

KLEUR [colour] Each bit of this memory location has its own meaning.

bit 0: If this bit is '0', the data of a character such as start address, length, tilting/rotating and/or shifting are determined from the table. If this bit is '1', we can set these data ourselves (special effects!!).

bit 1: If this bit is '0', XCOR is increased by the length of the character plus a fixed

number of positions according to memory location 'SPATIE'. In this way variable letter spacing is obtained (little space for the 'i', a lot of space for the 'm'). If this bit is '1', XCOR is increased by a fixed number (according to LTRAF). As a result all letters are placed at a fixed distance from each other.

bit 2: This bit determines whether the letters are placed on a white or a black background. '1' = black, '0' = white.

bits 3 and 4: These bits determine the 'colour' of the letter according to the table below:

4 3

0 0 not applicable ('read')

0 1 invert

1 0 white

1 1 black.

bit 5: If bit 5 is '1', a CR and LF are generated automatically at the end of a line (so we start on a new line).

bit 6: By making bit 6 '0', the background is not drawn. This means fewer dots have to be drawn per letter.

bit 7: By making bit 7 '0', the letter itself is not drawn but the background is. For special effects.

If bits 6 and 7 are both made 0, nothing happens on the screen. The program does, however, determine the start address and length of the character.

SPATIE [space] In this memory location the distance between two letters can be specified. Applies only if bit 1 of 'KLEUR' is zero (variable letter spacing).

LTRAF In this location a fixed letter spacing can be specified (normally = 6). All letters are at a fixed distance from each other. For LTRAF = 0, XCOR remains unchanged and all letters end up in the same place on the screen. Applies only if bit [1] of 'KLEUR' is one.

ASCII With this memory location we can specify which character we want to take from the character set. If bit 0 of 'KLEUR' is logic '1', we assume known data as regards length, start address, tilting/mirroring/shifting. In that case 'ASCII' is not looked at.

TBLHI and **TBLLO** These two memory locations give the start address of the table. Because none of the ASCII characters below 20 (hex) needs to be printed, we must reduce the start address by \$20. For the ASCII code \$22, for example, the 3rd number is then automatically taken from the table.

KARHI and **KARLO** These 2 locations give the 16-bit start address of the character set. b contains the x coordinate of the left margin. When a CR and LF are generated automatically (bit 5 of 'KLEUR' = 1), XCOR is set to the value of KNTLVO.

KNTLAC contains the x coordinate of the right margin. As soon as XCOR goes past this margin, either an LF and CR are generated, or XCOR is no longer increased. By setting this margin to FF we switch the margin off.

The settings above will seem somewhat confusing at first. For maximum flexibility the number of settings is certainly not excessive. In normal use all these settings, except ASCII, need to be made only once. The coordinates that determine where the letter

appears on the screen are always set automatically for the next letter after a letter has been drawn. This automatic behaviour can, incidentally, be switched off. For special applications all settings can be changed to your heart's content.

Optional settings

The memory locations 'LENGTE' [length], 'MODE', 'SUMLO' and 'SUMHI' contain data that are characteristic of a particular letter. In normal use (bit 0 of 'KLEUR' = 0) the program takes these data from the table. Bits 0...4 from the table indicate how many bytes a particular character occupies. This part is put in 'LENGTE'. Bits 5...7 indicate whether a character is rotated, mirrored and/or shifted. These control bits are put in 'MODE'. By adding the lengths of all preceding characters to the start address of the character set, the program determines at which address the character to be drawn begins. This address is put in SUMLO and SUMHI.

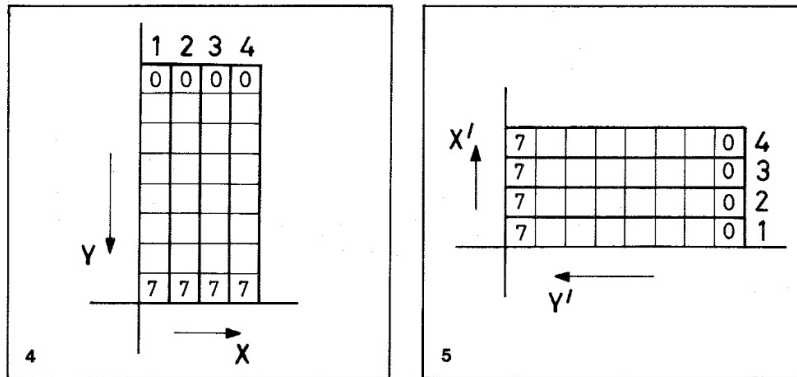
If bit 0 of 'KLEUR' = 1, the data above must be entered by the user. This can be particularly handy if, for a particular application, a single character is stored somewhere in memory. For the character set, the length of a character was limited to max. 31 bytes. Because the control bits are now separately in 'MODE', we can use the full 8 bits to specify the length. So for special characters that are not in the character set but somewhere 'loose' in memory, the maximum character size is 8 high x 255 wide, or 255 high x 8 wide. With symmetry this becomes 255 high x 16 long. Of course, independently of this, widening and heightening factors can still be applied.

Tilting/shifting and mirroring

In the table we can specify for each character whether it must be tilted, shifted and/or mirrored. If we do not use the character set (see 'Optional settings'), we can put these data in memory location 'MODE'. The codes are as follows:

7	6	5	
1	X	0	mirror middle leg double
1	0	1	mirror middle leg not double
X	1	X	rotate
0	X	1	shift

So with symmetrical characters we do not have the option of shifting. Bit 5 now indicates whether the last byte from the character set must be repeated when mirroring. You have already seen the effect in figs. 1 and 2 with the letters A and I. To show the effect of the various control codes clearly once more, we have applied all possible control-bit combinations to the letter y. You can see the result in fig. 3. The normal control code for this letter is 001 (shifted).



Auxiliary variables

To implement the program, several auxiliary variables have to be introduced. In fig. 3 you already see two of these auxiliary variables, namely YGRH and YGRL. If the background (also) has to be drawn, the program will fill all cells within these limits (where there are no letters) with the background colour. In this way all 'remnants' of old text disappear. YGRH lies 8 x the height factor above YCOR, and YGRL lies 2 x the height factor below YCOR. The height factor is the number in 'HOOG'. In normal use this will be 1. With control code '000' (normal situation, fig. 3) the letter is placed on the screen so that its bottom-left dot is at the coordinates (XCOR), (YCOR). If we have to shift (control code 001 or 011), YCOR is made equal to YGRL at the start of the program, so that the whole letter drops 2 dots. At the end of the program YCOR is set back to its original value. While the letter is being drawn, the two auxiliary variables XCOR2 and YCOR2 always indicate where the next dot is to be drawn. This is illustrated in fig. 4. First bit 7 of byte 1 is drawn. For each following bit YCOR2 is decremented by one (a lower Y coordinate is higher on the screen). When this process has been repeated 8 times, YCOR2 is set back to its initial value and XCOR2 is incremented by one. The following bytes are drawn in the same way. The number of bytes drawn corresponds to the number in 'LENGTE'. For a tilted character we would like to use the same program. That is possible by assuming the coordinate system of fig. 5. The direction of the coordinates is the same here as in fig. 4 (Y coordinate rising from bit 0 to 7, X coordinate rising from byte 1 to byte 4). So we see that not only are the X and Y coordinates swapped, but the directions are also reversed. The change of direction can be obtained by inverting the coordinates. In the normal situation we made XCOR2 equal to XCOR and YCOR2 equal to YCOR; now we must make XCOR2 equal to /YCOR and YCOR2 equal to /XCOR. When passing addresses to the display we must of course turn the coordinates back again. When the next bit is to be drawn, we decrement YCOR2 by one, just as before. Because YCOR2 is now in effect an inverted X coordinate, the final effect on the screen will be an increased X coordinate. As a result the new bit is drawn to the right of the previous bit. Symmetry is very easy to implement. When symmetry is required, a 'switch' is thrown on reaching the last byte to be drawn, so that not the next byte but the previous byte is taken from the character set. In this way (for a character length of 4) bytes 3, 2 and 1 are

drawn again after byte 4, which produces the mirror effect. With tilted symmetry (code 11X) the situation is somewhat more difficult. We now set the auxiliary coordinates as if the same character is to be placed once more tightly next to the character already drawn. This time, however, we shift the byte to be drawn not to the left but to the right. As a result bit 0, not bit 7, is drawn first. In this way the mirror image is produced in this case too.

Captions part 1:

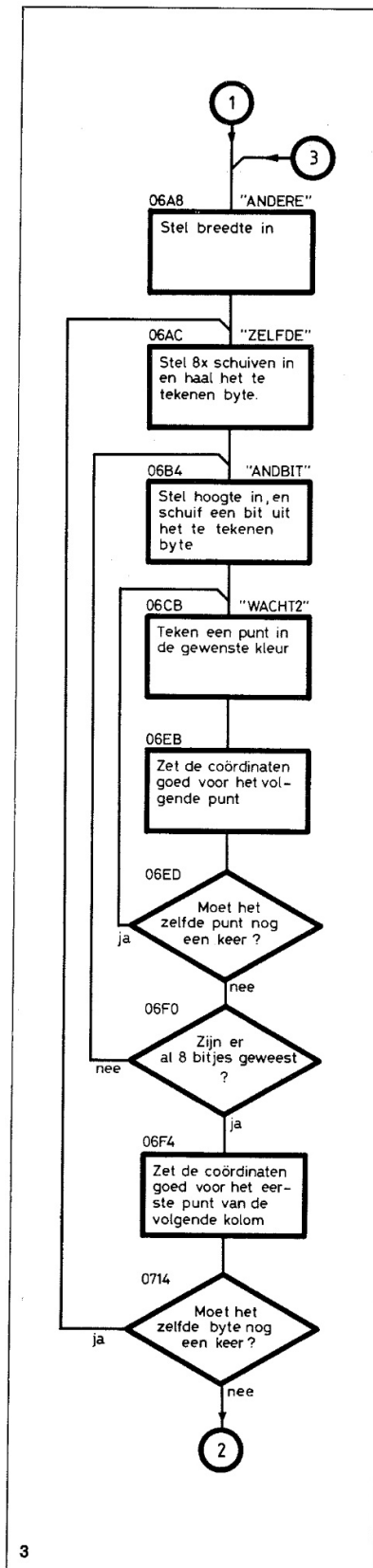
Fig. 1 This is how the characters are stored in the character generator. The last 5 bytes contain the characteristic data for the characters from the character set.

Fig. 2 This is the result on the screen.

Fig. 3 The effect of the 3-bit control code on the letter y.

Fig. 4 This is the normal direction of the X and Y coordinates.

Fig. 5 For rotated characters we assume a rotated coordinate system, so that the program runs in almost the same way.



remains unchanged during drawing and is used to set YCOR2 back to its old value after every 8 bits drawn.

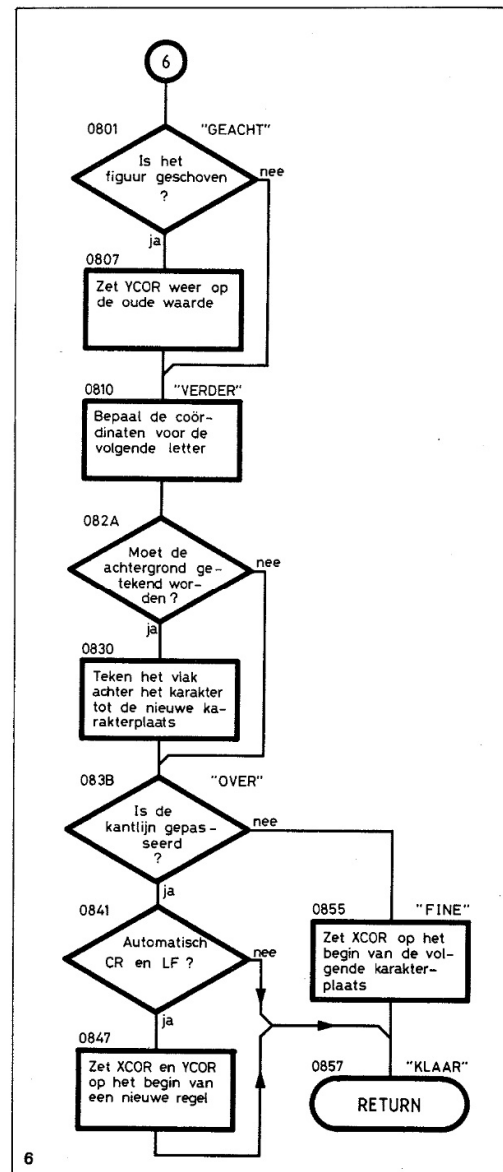
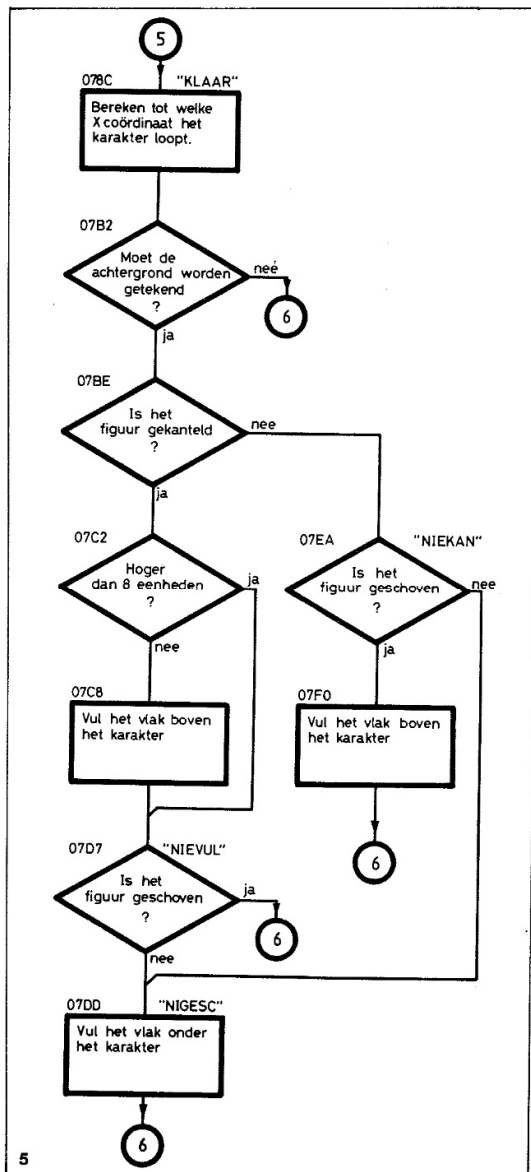
Drawing a character

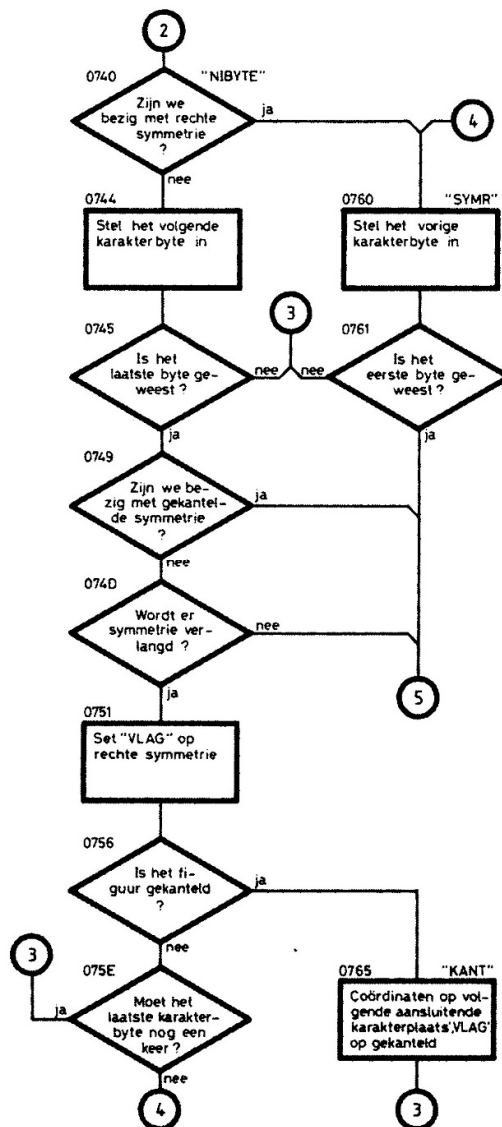
Figs. 3 and 4 show the flowchart of the core of the program. Fig. 3 gives the procedure for drawing one byte from the character set; fig. 4 makes sure that a new character byte is supplied each time. Because each character can be stretched both in width and in height, some loops and counters are needed. Each byte is placed vertically (assuming the normal situation), with the MSB at the bottom. If we want a character 3 times as wide, each byte must be drawn 3 times. That is why at the start of this program we must set a counter that keeps track of how many times the same byte has been drawn. This counter has been given the name 'BREED3'. In the next block (06AC) memory location 'TEL' [count] is set to 08. With this counter we keep track of whether all 8 bits of the byte have been drawn yet. Index register Y is used to point to the byte to be drawn and is now 0. At address 06B0 the first byte to be drawn is therefore fetched and put in 'TEMP'. The last counter to be set is the counter for the height factor. For this we use index register X. After all these settings we start shifting the first bit out of the byte to be drawn. Normally we shift to the left (MSB first). Only when we are working on the mirror image of a tilted figure must we shift to the right. Bit 6 of 'VLAG' [flag] indicates this situation. In any case, after the shift the bit to be drawn is in the carry.

Draw a dot in the desired colour

We now come to the heart of all the loops (fig. 3). Because this part of the program is repeated a great deal, it was designed for speed rather than for a small number of program steps. When drawing one dot, the following situations can occur: the dot must be drawn in the foreground colour (carry=1); the dot must be drawn in the background colour (carry=0); the dot must not be drawn (bit 6 or 7 of 'KLEUR' = 0 and carry =

0 or 1 respectively). In the first two cases the colour is set, after which there is a jump to address 06CB. Here only the new Y address is clocked in, because the X address remains unchanged while a byte is being drawn. (For a tilted character, of course, only the X address is clocked in.) Then the write pulse is given and the dot has been drawn. If the dot was not to be drawn, because bit 6 or 7 (6 for the background and 7 for the foreground) was '0', YCOR2 is set so that it points to the position of the next bit to be drawn. In all cases the height factor (at this moment in index register X) indicates how many times a dot must be repeated before the next bit is shifted out of the byte being drawn. This is shown by the innermost loop (fig. 3). Each time this loop is run, register X is decremented by one. Only when this register is 0 is it checked, at address 06F0, whether another bit of this byte must be drawn. As long as 'TEL' (which we had set to 8) is not yet '0', the byte to be drawn is shifted once more.





4

checked whether the counter 'BREED3' has reached 0 yet. As long as this is not the case, the whole preceding procedure is repeated, one position further on the screen. As soon as a new byte is needed we arrive in fig. 4.

Determining the next byte

Fig. 4 shows the part of the flowchart that not only determines which byte is to be drawn next, but also performs all the actions needed to mirror a character (symmetry). We distinguish between 'straight' symmetry (as with A and I, see the previous part) and 'tilted' symmetry (as with the chess rook).

Of course the height factor is also set again, because the new bit may also have to be drawn a number of times one above the other. When the byte has been 'shifted empty', we must set the coordinates right again for the next column. This happens from address 06F4. Here YCOR2 gets its original value back, while XCOR2 is incremented by one. Then the X and Y addresses are put in the display without generating a write pulse. For characters in the normal position only the X address needs to be clocked in (XCOR2 X address), because the Y address is clocked in again for every bit. For a rotated character it is just the other way round: here XCOR2 must be put in the Y buffer, while the X address is clocked in again for every bit. We now arrive at the last loop in fig. 3. The same byte must be drawn as many times as the widening factor indicates. At address 0714 it is

Straight symmetry is indicated by bit 7 of 'VLAG', tilted symmetry by bit 6. We enter fig. 4 at address 0740. As long as not all bytes from the character set have been drawn, the next byte from the character set is set up at address 0744. After that we jump back to fig. 3, where the new byte is drawn at the right size. As soon as the last byte has been drawn, we arrive via 0749 at address 074D, where it is checked whether the character must be mirrored. If not, we roll on to fig. 5. If symmetry is requested, we provisionally set bit 7 of 'VLAG' to indicate straight symmetry. At 0756 it is now checked whether the figure is really upright. For now we assume that it is. The last conditional jump is found at 075E. Here it is checked whether the middle leg must be taken double, as for example with the A, or single, as for example with the I (see the previous part). With a double middle leg we jump back again to fig. 3, where the whole process is repeated in exactly the same way. The only difference from the first time is that bit 7 of 'VLAG' is now logic 1. As a result the 'switch' at address 0740 (at the top of fig. 4) has been thrown, and the previous character byte is set up each time. This process continues until the 1st byte of the character from the character set has been drawn. At that moment the mirrored part is also finished and we go to fig. 5. In the last case, tilted symmetry, we go at 0756 (fig. 4) to 0765. Here the auxiliary coordinates are set to the next character position, but without spacing. Memory location 'VLAG' is now set to tilted symmetry, and register Y is made 0, so that it points to the first character byte again. Because 'VLAG' is set to tilted symmetry, the switch at address 0740 is not operated now (so the program simply takes the next byte each time). The character byte is, however, always shifted to the right instead of to the left, so that bit 0 is drawn first. When the mirror image has been produced in this way, we go via 0745 and 0749 to fig. 5.

Filling the space between the letters

The most important work has been done; the letter has been drawn. We are not there yet, however: because of the possibilities of tilting and shifting, not all letters have the same height. If we want to draw the background, all pixels between YGRL and YGRH where there is no letter must get the background colour. Only then are all remnants of old text and/or drawings overwritten (see fig. 3 of the previous part). The background is filled in according to fig. 5.

Only tilted characters can be higher than 8 dots. In that case the space above the letter cannot be filled (07C2).

Fig. 6 shows the last part of the flowchart. Here the coordinates of the next letter position are determined. Only when this is known is the space behind the letter filled up to the next letter position.

Captions part 2:

Fig. 1 The connections to the PIA.

Fig. 2 The first part of the flowchart. Here the various memory locations get their initial values.

Fig. 3 This part of the flowchart draws one byte at the desired height and width.

Fig. 4 The part that takes care of mirroring the character, if required.

Fig. 5 With this part of the flowchart the space below and above the letters is drawn in the background colour.

Fig. 6 The last part. Here the next letter position is calculated.

Listing 1 The program. The addresses printed in bold are adapted to the I/O of the KIM and must be adapted for other systems. If a different start address is wanted, the instructions at 07D4, 07E5, 07FE and 0838 must be changed.

Opdracht	Opdrachtnummer	Opdrachtnaam	Opdrachthandelingen	Opdrachtomschrijving
0600	D8	START	CLD	
0601	A5 A5		LDA KLEUR	
0603	29 18		AND #S18	
0605	85 BA		STA VOOR	
0607	A5 A5		LDA KLEUR	
0609	29 04		AND #S04	Initialiseren
060B	0A		ASL	
060C	09 10		ORA #S10	
060E	85 BB		STA ACHTER	
0610	A2 00		LDX #S00	
0612	86 B2		STX VLAG	
0614	A5 A5		LDA KLEUR	
0616	29 01		AND #S01	Gegevens bekend?
0618	D0 2C		BNE BEKEND	Zo ja, naar 'BEKEND'.
061A	A5 D2		LDA KARLO	
061C	85 C0		STA SUMLO	Kopieer het
061E	A5 D3		LDA KARHI	beginadres.
0620	85 C1		STA SUMHI	
0622	A4 A0		LDY ASCII	
0624	B1 D4		LDA (TBLL0), Y	
0626	29 E0		AND #SE0	Stel 'MODE' in.
0628	85 C2		STA MODE	
062A	B1 D4		LDA (TBLL0), Y	
062C	29 1F		AND #S1F	Stel 'LENGTE' in.
062E	85 C3		STA LENGTE	
0630	88	TELOP	DEY	
0631	C0 1F		CPY #S1F	Eén karakter terug.
0633	F0 11		BEQ BEKEND	Indien laatste
0635	B1 D4		LDA (TBLL0), Y	karakter naar 'BEKEND'.
0637	29 1F		AND #S1F	Karakterlengte
0639	18		CLC	uit de tabel.
063A	65 C0		ADC SUMLO	
063C	85 C0		STA SUMLO	
063E	A9 00		LDA #S00	Tel alle lengten bij elkaar
0640	65 C1		ADC SUMHI	op om het startadres van
0642	85 C1		STA SUMHI	het karakter te vinden.
0644	90 FA		BCC TELOP	Onvoorwaardelijk terug.
0646	A0 00		LDY #S00	
0648	A5 A2		LDA YCOR	
064A	85 BC		STA YGRL	
064C	A6 A4		LDX HOOG	
064E	38	LOOP	SEC	
064F	E9 08		SBC #S08	
0651	E6 BC		INC YGRL	Bepaald YGRL en YGRHI.
0653	E6 BC		INC YGRL	
0655	CA	DEX		
0656	D0 F6		BNE LOOP	
0658	85 BD		STA YGRHI	
065A	A9 20		LDA #S20	
065C	24 C2		BIT MODE	
065E	F0 06		BEQ VERDER	Indien geschoven
0660	30 04		BMI VERDER	YCOR goed zetten.
0662	A5 BC		LDA YGRL	
0664	85 A2		STA YCOR	
0666	AD 02 17	WACHT1	STA STUUR	Wacht op syncpuls
0669	10 FB		BPL WACHT1	van de TV.
066B	A5 A2		LDA XCOR	
066D	8D 00 17		STA ADRES	
0670	EE 02 17		INC STUUR	
0673	EE 02 17		INC STUUR	X en Y-adres in het
0676	A5 A2		LDA YCOR	display zetten.
0678	8D 00 17		STA ADRES	
067B	8E 02 17		STX STUUR	
067E	A5 A1		LDA XCOR	
0680	85 88		STA XCOR2	
0682	A5 A2		LDA YCOR	
0684	85 86		STA YCOR2	
0686	85 87		STA YCOR3	Hulpcoördinaten
0688	A5 A3		LDA BREED	instellen voor
068A	85 B4		STA BREED2	rechtstaand figuur
068C	A5 A4		LDA HOOG	
068E	85 83		STA HOOG2	
0690	50 16		BVC ANDERE	Spring indien niet
				gedraaid
0692	A5 A1		LDA XCOR	
0694	49 FF		EOR #SFF	
0696	85 86		STA YCOR2	
0698	85 87		STA YCOR3	
069A	A5 A2		LDA YCOR	
069C	49 FF		EOR #SFF	Hulpcoördinaten
069E	85 88		STA XCOR2	instellen voor
06A0	A5 A3		LDA BREED	gekanteld figuur
06A2	85 83			

06AA	85	B5		STA	BREED3	Opnieuw in.
06AC	A9	B8	ZELFDE	LDA	#S08	Stel 8x
06AE	A5	B9		STA	TEL	schuiven in.
06B0	B1	C0		LDA	(SUMLO),	Y
06B2	85	B1		STA	TEMP	Zet het te tekenen
06B4	A6	B3	ANDBIT	LDA	HOOG2	Stel hoogte in.
06B6	24	B4		BIX	VLAG	
06B8	70	A4		BVS	RECHTS	Schuif een bit
06BA	06	B1		ASL	TEMP	uit 'TEMP'
06BC	50	02		BVC	NORM	Normaal naar links
06BE	46	B1	RECHTS	LSR	TEMP	
06C0	24	A5	NORM	BIT	KLEUR	Voor achtergrondkleur
06C2	90	5A		BCC	ACHTGR	naar 'ACHTGR'.
06C4	10	61		BPL	NIET	Voorg. tekenen?
06C6	A5	BA		LDA	VOOR	
06C8	8D	02 17		STA	STUUR	Stel voorgrond in
06CB	AD	02 17	WACHT2	LDA	STUUR	Wacht op synclups
06CE	10	FB		BPL	WACHT2	van de TV.
06D0	24	C2		BIT	MODE	Normaal naar 'YKLOK'
06D2	50	5A		BVC	YKLOK	om Y in te klokken.
06D4	A9	FF		LDA	#\$FF	
06D6	45	B6		EOR	YCOR2	
06D8	8D	00 17		STA	ADRES	Klok het X-adres in.
06DB	EE	02 17		INC	STUUR	
06DE	A9	02		LDA	#S02	
06E0	0D	02 17		ORA	STUUR	
06E3	8D	02 17		STA	STUUR	Schrijfpuls
06E6	29	18	RUST	AND	#\$8	Terug naar de
06E8	8D	02 17		STA	STUUR	rusttoestand
06EB	C6	B6		DEC	YCOR2	Als hetzelfde punt
06ED	CA			DEX		nog een keer moet,
06EE	D0	DB		BNE	WACHT2	naar 'WACHT2'.
06F0	C6	B9	NIBIT	DEC	TEL	Zijn we al bij het
06F2	D0	C0		BNE	ANDBIT	laatste bitje?
06F4	A5	B7		LDA	YCOR3	
06F6	85	B6		STA	YCOR2	Zet de coördinaten goed
06F8	E6	B8		INC	XCOR2	voor de volgende kolom.
06FA	AD	02 17	WACHT3	LDA	STUUR	Wacht op synclups
06FD	10	FB		BPL	WACHT3	van de TV.
06FF	A5	B8		LDA	XCOR2	
0701	8D	00 17		STA	ADRES	Klok het nieuwe
0704	EE	02 17		INC	STUUR	X-adres in.
0707	EE	02 17		INC	STUUR	
070A	A9	FF		EOR	#\$FF	
070C	8D	00 17		STA	ADRES	Klok het gedraaide
070F	A9	00		LDA	#S00	X-adres in.
0711	8D	02 17		STA	STUUR	
0714	C6	B5		DEC	BREED3	Als hetzelfde byte nog
0716	D0	94		BNE	ZELFDE	'n keer moet: naar 'ZELFDE'.
0718	F0	26		BEQ	NIBYTE	Onvoorwaardelijk
071A	F0	8C	ANDER2	BEQ	ANDERE	Tussensprong voor
071C	D0	8A		BNE	ANDERE	long branches
071E	50	07	ACHTGR	BVC	NIET	Achterg. tekenen?
0720	A5	BB		LDA	ACHTER	Stel achtergrond in.
0722	8D	02 17		STA	STUUR	
0725	70	A4		BVS	WACHT2	Onvoorwaardelijk
0727	C6	B6	NIET	DEC	YCOR2	Verplaats de pointer
0729	CA			DEX		zonder te tekenen.
072A	D0	FB		BNE	NIET	
072C	F0	C2		BEQ	NIBIT	Onvoorwaardelijk
072E	A5	B6	YKLOK	LDA	YCOR2	
0730	8D	00 17		STA	ADRES	Klok het Y-adres
0733	A9	02		LDA	#S02	in.
0735	0D	02 17		ORA	STUUR	
0738	8D	02 17		STA	STUUR	
073B	EE	02 17		INC	STUUR	Schrijfpuls
073E	D0	A6		BNE	RUST	Onvoorwaardelijk
0740	24	B2	NIBYTE	BIT	VLAG	'Bij spiegelen verder
0742	30	1C		BMI	SYMR	op 'SYMR'.
0744	C8			INY		Volgende byte
0745	C4	C3		CPY	LENGTE	Zolang laatste byte
0747	D0	D1		BNE	ANDER2	niet is geweest: terug.
0749	24	B2		BIT	VLAG	Als we klaar zijn met
074B	70	3F		BVS	KLAAR	spiegelen, of als er
074D	C4	C2		BIT	MODE	geen symmetrie gevraagd
074F	10	3B		BPL	KLAAR	wordt naar 'KLAAR'.

0763 30 27	BMI KLAAR	Onvoorwaardelijk
0765 A5 B7	KANT LDA YCOR3	
0767 A6 A3	LDX BREED2	
0769 38	SEC	
076A E9 08	COR SBC #S08	Coördinaten instellen op volgende aansluitende karakterplaats.
076C CA	DEX	
076D D0 FB	BNE COR	
076F 85 B6	STA YCOR2	
0771 85 B7	STA YCOR3	
0773 A5 A2	LDA YCOR	
0775 8D 00 17	STA ADRES	
0778 A2 02	LDX #S02	Nieuwe adressen inklokken.
077A 8E 02 17	STX STUUR	
077D A0 00	LDY #S00	
077F 8C 02 17	STY STUUR	
0782 49 FF	EOR #SFF	
0784 85 B8	STA XCOR2	XCOR2 goed zetten
0786 A9 40	LDA #S40	
0788 85 B2	STA VLAG	Zet 'VLAG' op gekantelde symmetrie
078A D0 8E	BNE ANDER2	Onvoorwaardelijk
078C A5 C3	KLAAR LDA LENGTE	
078E 85 B1	STA TEMP	Lengte in 'TEMP'
0790 24 C2	BIT MODE	
0792 50 06	BVC TEST	
0794 08	PHP	
0795 A9 08	LDA #S08	Indien gekanteld: karakterlengte is 8
0797 85 B1	STA TEMP	
0799 28	PLP	
079A 10 0A	TEST BPL GEWOON	Is er symmetrie?
079C 06 B1	ASL TEMP	Bij symm. lengte x 2
079E A9 20	LDA #S20	
07A0 24 C2	BIT MODE	
07A2 F0 02	BEQ GEWOON	Bij enkele middenpoot lengte - 2
07A4 C6 B1	DEC TEMP	
07A6 A6 A3	GEWOON LDX BREED	
07A8 A5 A1	LDA XCOR	
07AA 18	LOOP1 CLC	
07AB 65 B1	ADC TEMP	Bereken de nieuwe X-coördinaat, en zet hem in 'TEMP'
07AD CA	DEX	
07AE D0 FA	BNE LOOP1	
07B0 85 B1	STA TEMP	
07B2 24 A5	BIT KLEUR	
07B4 50 4B	BVC GEACHT	
07B6 A5 A1	LDA XCOR	
07B8 85 B8	STA XCOR2	Coördinaten van het te vullen vlak.
07BA A5 BD	LDA YGRH	
07BC 85 B7	STA YCOR3	
07BE 24 C2	BIT MODE	
07C0 50 28	BCV NIEKAN	Indien niet gekanteld naar 'NIEKAN'
07C2 A5 C3	LDA LENGTE	
07C4 C9 08	CMP #S08	Indien hoger dan 8 maar 'NIEVUL'
07C6 B0 0F	BCS NIEVUL	
07C8 A5 A2	LDA YCOR	
07CA A6 A4	LDX HOOG	
07CC 38	SEC	
07CD E5 C3	LOOP2 SBC LENGTE	Bereken de gekantelde hoogte
07CF CA	DEX	
07D0 D0 FA	BNE LOOP2	
07D2 85 B6	STA YCOR2	
07D4 20 58 08	NIEVUL JSR TKNBLK	Teken boven het kar.
07D7 A5 BC	LDA YGRL	Indien niet geschoven strook onder het karakter tekenen.
07D9 C5 A2	CMP YCOR	
07DB F0 24	BEQ GEACHT	
07DD A5 BC	NIGESC LDA YGRL	
07DF 85 B6	STA YCOR2	
07E1 A5 A2	LDA YCOR	Vul de strook onder het karakter.
07E3 85 B7	STA YCOR3	
07E5 20 58 08	JSR TKNBLK	
07E8 F0 17	BEQ GEACHT	Onvoorwaardelijk
07EA A5 BC	NIEKAN LDA YGRL	Indien niet geschoven onderstrook gaan tekenen.
07EC C5 A2	CMP YCOR	
07EE D0 ED	BNE NIGESC	
07F0 A5 A4	LDA HOOG	
07F2 0A	ASL	
07F3 0A	ASL	Bereken karakterhoogte.
07F4 0A	ASL	
07F5 85 B3	STA HOOG2	
07F7 38	SEC	
07F8 A5 A2	LDA YCOR	
07FA E5 B3	SBC HOOG2	Vul de strook boven het karakter.
07FC 85 B6	STA YCOR2	
07FE 20 58 08	JSR TKNBLK	
0801 A5 BC	GEACHT LDA YGRL	
0803 C5 A2	CMP YCOR	Was het karakter geschoven?
0805 D0 09	BNE VERDER	
0807 A6 A4	LDX HOOG	
0809 C6 A2	LOOP3 DEC YCOR	

080B C6 A2	DEC YCOR	Zet YCOR weer op de oude waarde.
080D CA	DEX	
080E D0 F9	BNE LOOP3	
0810 A5 B1	VERDER LDA TEMP	
0812 85 B8	STA XCOR2	Bereken nieuwe letterplaats voor variabele letterafstand
0814 18	CLC	
0815 65 A6	ADC SPATIE	
0817 85 B1	STA TEMP	
0819 A9 02	LDA #S2	Moet er vaste letterafstand ingesteld worden?
081B 24 A5	BIT KLEUR	
081D F0 09	BEQ VARI	
081F 08	PHP	
0820 A5 A1	LDA XCOR	Bereken de nieuwe letterplaats voor vaste letterafstand
0822 18	CLC	
0823 65 A7	ADC LTRAF	
0825 85 B1	STA TEMP	
0827 28	PLP	
0828 A5 B8	VARI LDA XCOR2	
082A C5 B1	CMP TEMP	Ruimte tussen de letters opvullen?
082C B0 D	BCS OVER	
082E 50 0B	BVC OVER	
0830 A5 BC	LDA YGRL	
0832 85 B6	STA YCOR2	Vul het vlak vanaf laatste karakter tot de nieuwe letterplaats.
0834 A5 BD	LDA YGRH	
0836 85 B7	STA YCOR3	
0838 20 58 08	JSR TKNBLK	
083B A5 B1	OVER LDA TEMP	
083D C5 D0	CMP KNTLAC	Is de achter-kantlijn gepasseerd?
083F 90 14	BCC FINE	
0841 A9 20	LDA #S20	
0843 24 A5	BIT KLEUR	Automatisch CR en LF?
0845 F0 10	BEQ KLAAR	
0847 A6 A4	LDX HOOG	
0849 A5 A2	LDA YCOR	
084B 18	LOOP4 CLC	
084C 69 0A	ADC #S0A	Verzorg een LF.
084E CA	DEX	
084F D0 FA	BNE LOOP4	
0851 85 A2	STA YCOR	
0853 A5 D1	LDA KNTLVO	Verzorg een CR
0855 85 A1	FINE STA XCOR	
0857 60	KLAAR RTS	Terug naar hoofdprog.
0858 A5 BB	TKNBLK LDA ACHTER	
085A 8D 02 17	STA STUUR	Stel begincoördinaten in.
085D A4 B6	LDY YCOR2	
085F A6 B8	LDX XCOR2	
0861 AD 02 17	LOOP5 LDA STUUR	
0864 10 FB	BPL LOOP5	
0866 8E 00 17	STX ADRES	
0869 EE 02 17	INC STUUR	
086C EE 02 17	INC STUUR	
086F 8C 00 17	STY ADRES	Vul een vlak, begrensd door XCOR2, TEMP YCOR2 en YCOR3.
0872 EE 02 17	INC STUUR	
0875 8D 02 17	STA STUUR	
0878 E8	INX	
0879 E4 B1	CPX TEMP	
087B D0 E4	BNE LOOP5	
087D 88	DEY	
087E C4 B7	CPY YCOR3	
0880 D0 DD	BNE LOOP5	
0882 60	RTS	Terug.

The (zero page) locations used can be divided into four groups:

Program constants

These are variables that normally only need to be set once, at the start of the whole program.

00D0 KNTLAC Margin front and

00D1 KNTLVO rear

00D2 KARLO Start address of the

00D3 KARHI character set

00D4 TBLLO Start address of the table (belonging to

00D5 TBLHI the character set) –\$20

[Note: according to part 1, KNTLVO is the left (front) margin and KNTLAC the right (rear) margin.]

Control numbers

These variables directly affect the way a character appears on the screen.

00A0 ASCII indicates which character

00A1 XCOR X coordinate of the character

00A2 YCOR Y coordinate of the character

00A3 BREED Widening factor

00A4 HOOG Heightening factor

00A5 KLEUR Eight control bits (see RB May 1979)

00A6 SPATIE Space between two letters (variable spacing)

00A7 LTRAF The distance between the start of one letter and the start of the next letter (fixed spacing)

Character constants

These variables can be determined either by the subroutine itself or (for special effects) by another program. In the latter case 'ASCII' is not looked at.

00C0 SUMLO start address of the

00C1 SUMHI character to be drawn

00C2 MODE Tilt/shift/mirror?

00C3 LENGTE Number of bytes the character needs in the character set.

Internal variables

These memory locations are only used while the program is running.

00B1 TEMP Temporary register

00B2 VLAG Set for the 2nd half of symmetry

00B3 HOOG2 \

00B4 BREED2 |

00B5 BREED3 | Auxiliary coordinates for tilting
00B6 YCOR2 | and restoring initial values
00B7 YCOR3 |
00B8 XCOR2 /
00B9 TEL Number of bits to be shifted out
00BA VOOR Foreground colour (black, white, invert)
00BB ACHTER Background colour (black, white)
00BC YGRL Lower limit of text bar
00BD YGRH Upper limit of text bar

CORRECTION

Unfortunately a small error crept into listing 1:
at address 06B7 it should read B2 instead of B4 ('VLAG').

Table Character map

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