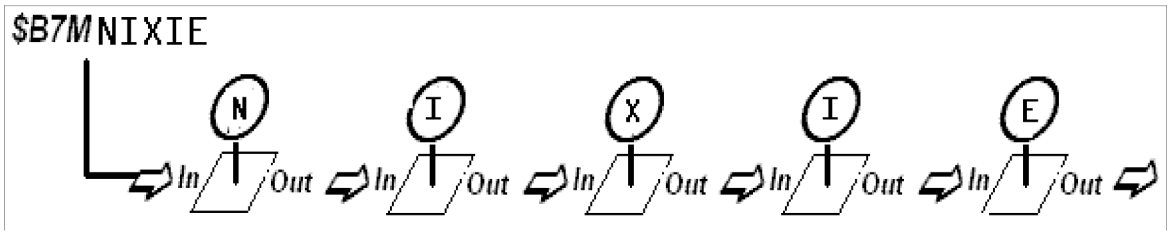
Smart Sockets

Through the use of simple ASCII type instructions it is possible to use this device to produce arrays of many display devices, of similar and different types. The device features 8 built in numerical fonts, 2 special fonts for use with nixie tubes and bargraph displays, and 10 transition effects. The standard Burroughs alphanumeric font is pre-programmed and in addition to this there are 124 user definable characters. The industry standard 9600 Baud, 8 data bits, No parity,1 stop bit , 8N1, protocol enables almost any computer, handheld PDA, or microcontroller to communicate with the display elements directly, allowing for rapid development, and enabling some complex display routines to be generated from very simple commands.

General Information

It is not unusual to want to generate complex display routines, only to find that you run out of memory, or have reached limits of performance of the controlling devices such that progress is hampered and compromises have to be made. The idea behind this device came from a need to generate complex display routines through the use of a microcontroller. Having good programming skills is usually an advantage in such situations but even then it can become difficult to make a display which can respond to variable data in addition to the host performing the other functions which it has been assigned to do. From a hardware point of view it is also convenient to mount all of the driving components on the same board as the B7971. With the exception of the power supply and data lines, there are no other connections required to make these boards produce the type of display which meets the modern expectations of complexity and visual interest. Each cathode has its own current limiting resistor fitted, in accordance with the original Burroughs recommended best practice. This means that a constant light output is guaranteed from each segment to make a uniform display regardless of how many segments are illuminated. The 'smart' aspect of the device comes from the way in which each socket communicates with its neighbour in order to establish its own position in the display array. This means that if you send an ascii text string such as ' NIXIE ' to a display array of 5 sockets each character is displayed on the correct display with respect to it's position in the text string. So the first tube displays the 'N' , the next tube the 'I', and so on...

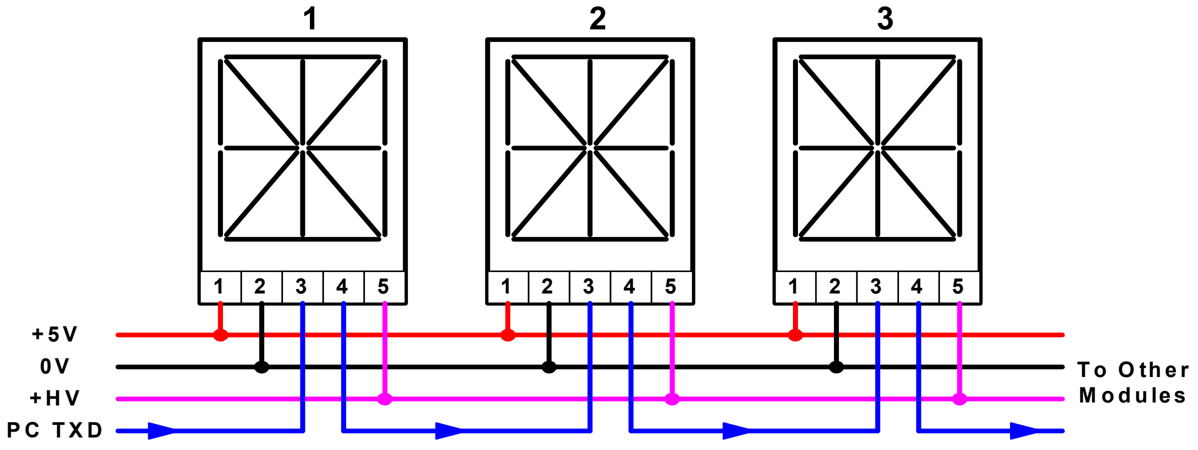


A flexible data input format allows for the characters, fonts, effects and the speed of those effects to be changed in ways which suit different programming constructs and the different types of terminal programs which can be used to control the sockets.

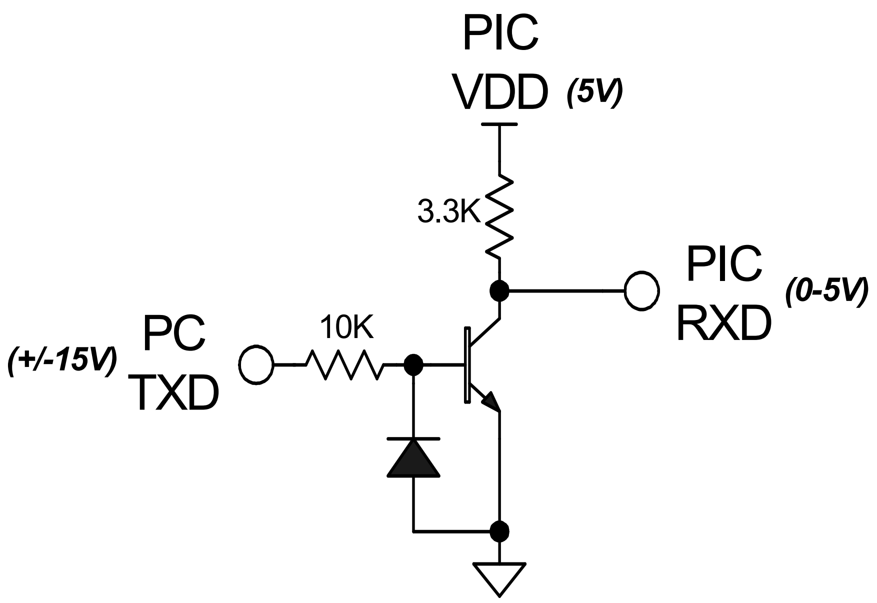
Advanced Programming

With some clever programming, it would be possible for the host controller to properly parse the string being sent so that it will fit on the display correctly without overflowing. In order to do this the host will need to know how many display elements or smart sockets make up the display array. It is possible for the host to read the value being transmitted by the last socket in the array at startup. Doing this will provide a binary value of the number of sockets in the array. At startup the first socket assigns a value of 1 to itself and communicates with its downstream neighbour such that the next socket in linecalls itself number 2…and so on down the line. This only happens once during each startup cycle, within 100mS of power being applied.

Module Interconnection



The serial data level is TTL compatible with idle state = 1 which makes driving arrays of sockets with a microcontroller a very simple task. If you wish to use a PC, or any other device that transmits RS232 data levels the use of a level shifter is recommended. One of the most common is the MAX232 device. Many equivalents are also available. The MAX233 does not require the use of additional capacitors at all, which is a bonus, and the MAX232A requires you to use only 4 capacitors with a value of 0.1uF. You do not need to use a MAX232 or equivalent, a simple level shift arrangement can be achieved using the following circuit.



PC RS232 to PIC Level Shift Circuit

The Instruction Set

Instructions are always sent to the first socket in an array. Each socket in turn resends the instructions to the following socket and extracts any useful information intended for itself. Each socket is aware of its position in an array such that it can calculate which character in a string of characters is to be interpreted All instructions take the following form:

```
<3CharHeader>+<1CharCommand>+<Command Values>+<cr>
```

The header and the command characters must all be in upper case. (Replace <cr> with ascii character 13 where necessary.) The three character header is always = $B7, which allows the sockets to be connected to an existing data bus with minimal risk of it being affected by an existing flow of data . Message validity is checked upon the receipt of the final <cr>.

Commands

There are two types of commands. Display commands and resource commands.

DISPLAY COMMANDS

M	Group **m**essage string
U	Set the **u**nderscore segment on or off at any position in the array

[M]essage display

```
$B7Mxxxxxxxx<cr>
```

Where xxxxxx is the string to display. The group message command is the most common message in most cases. With this command you are transmitting the actual display data to the array of display devices.

If you wish to display the string ' NIXIE' across the array (Note there are no spaces or comma's):
$B7MNIXIE<cr>

For fonts 0-9
Available characters are "0" to "9", "A" to "Z", Space and Del (ASCII 8). All other characters will be ignored except <cr> and "!" which are special control characters.

For user defined font U
Available characters for display and remapping are ASCII 0 to 125 (Null to "}"). As with fonts 0-9, <cr> and "!" are special control characters and may not be remapped.

[U]nderscore display
$B7Uxxxxxxxxx<cr>

Where 1 = underscore on in that position, 0 turns off the underscore for that position, and 'x' can be any other character. Onl;y 0's and 1's are responded to when the underscore command has been sent. Underscores are not turned on immediately but their requested status either on or off is remembered for the next time a character change command is sent to the same position. For example , sending $B7U1!!!!! will instruct the socket to turn on the underscore at position 1 at the next character transition, so afterwards if you send $B7M8, then the first position will display the character '8' and the underscore will come on at the same time The underscore inherits the transition effect in use for the characters. Where there is a user defined character being displayed which makes use of the underscore, any instructions to set or clear the underscore are ignored. The null character can be used with fonts and effects just as in the same way as it is used in character messages. In this way a new command can be transmitted by the host controller without it needing to have remembered the current settings in order to overwrite them correctly.

To turn on the underline in digit 4 and off in digit 5 and leaving all other digits unchanged:
$B7U!!!10!!!!<cr>

The Null Character '!'
An important character is the exclamation mark " ! " When this character is used in a command it is interpreted as 'no instruction' to the socket of the same relative position as the null character has in the string. This is useful in a message because you can send an instruction that only changes one character and leaves the existing display unchanged. The null character can also be used in the resource commands for Font, Effect and Speed.

To set the character in the 4th position to the letter 'S' and leave the rest of the display untouched:
$B7M!!!S!!!!!!!<cr>

RESOURCE COMMANDS
E Set individual **e**ffects

F	Set individual **f**onts
S	Set individual effect **s**peeds
W	**W**rite a bit pattern to an eeprom location to create a user defined character

Resource commands are a useful way to set the fonts, effects and the speed of the effect used at the next transition. The current display is not altered during the programming of effects, fonts and speeds, which allows them to be sent 'silently' in the background. Even if a transition or effect is in progress the new resource commands will still be received and the new values stored for the next transition. A different font can be assigned to each position in the array, should it be desired. Also, different transition effects and effect speeds can be run in any character position.

[E]ffects command

`$B7Exxxxxx<cr>`

Where xxxxxx are the numbers of the new effects, relative to their position in the display array. Valid effect numbers are 0 to 9.

To change the effects of the second and fourth positions to 5 and all other positions to effect 2:
`$B7E252522<cr>`

The Effects

0 = No effect, instant change
1 = Cross-fade
2 = Jump Fade (instant on, fade out)
3 = Fade out, fade in
4 = Zipper, bilateral downward wipe and refresh
5 = Shifter, characters wiped left to right *
6 = Segment deconstruction and rebuild
7 = Spin, half cycle
8 = Spin, full cycle
9 = Radar

*Used sequentially this can give the effect of a wipe across all tubes starting at the left and moving to the right.

[F]onts command

`$B7Fxxxxxx<cr>`

Where xxxxxx are the new fonts to be used, relative to their position in the display array. Valid font numbers are 0 to 9 and ' U '.

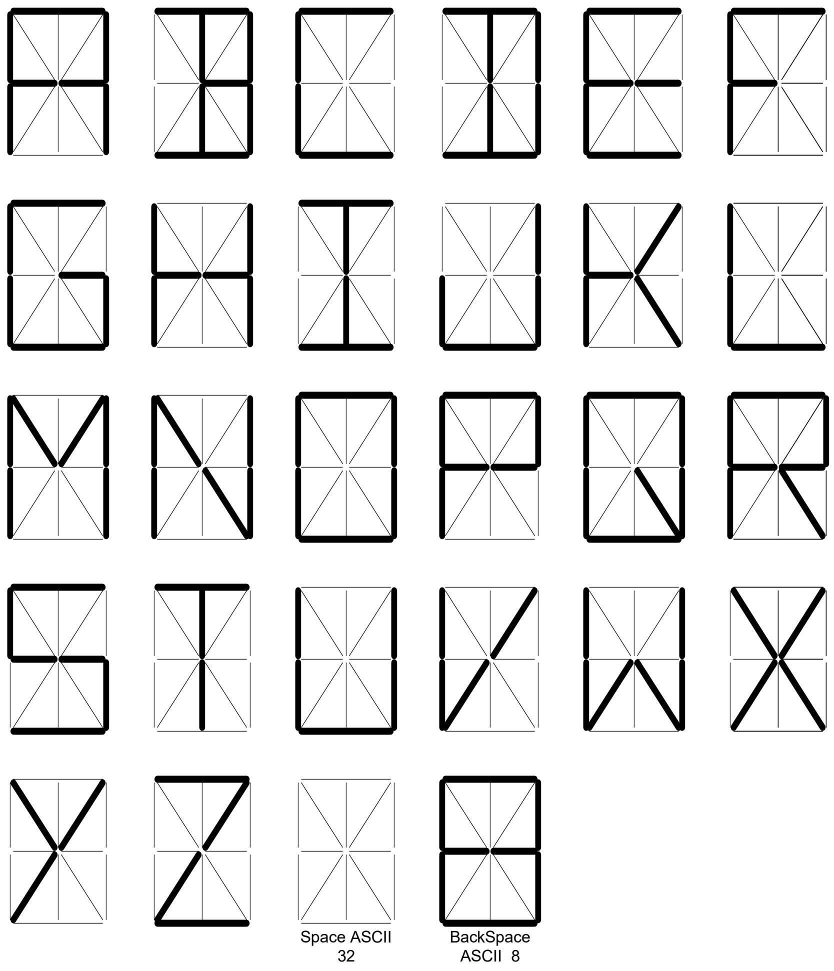
To change the fonts of the first and third tubes to font 3, and all other positions to font 1:
`$B7F313111<cr>`

The Fonts

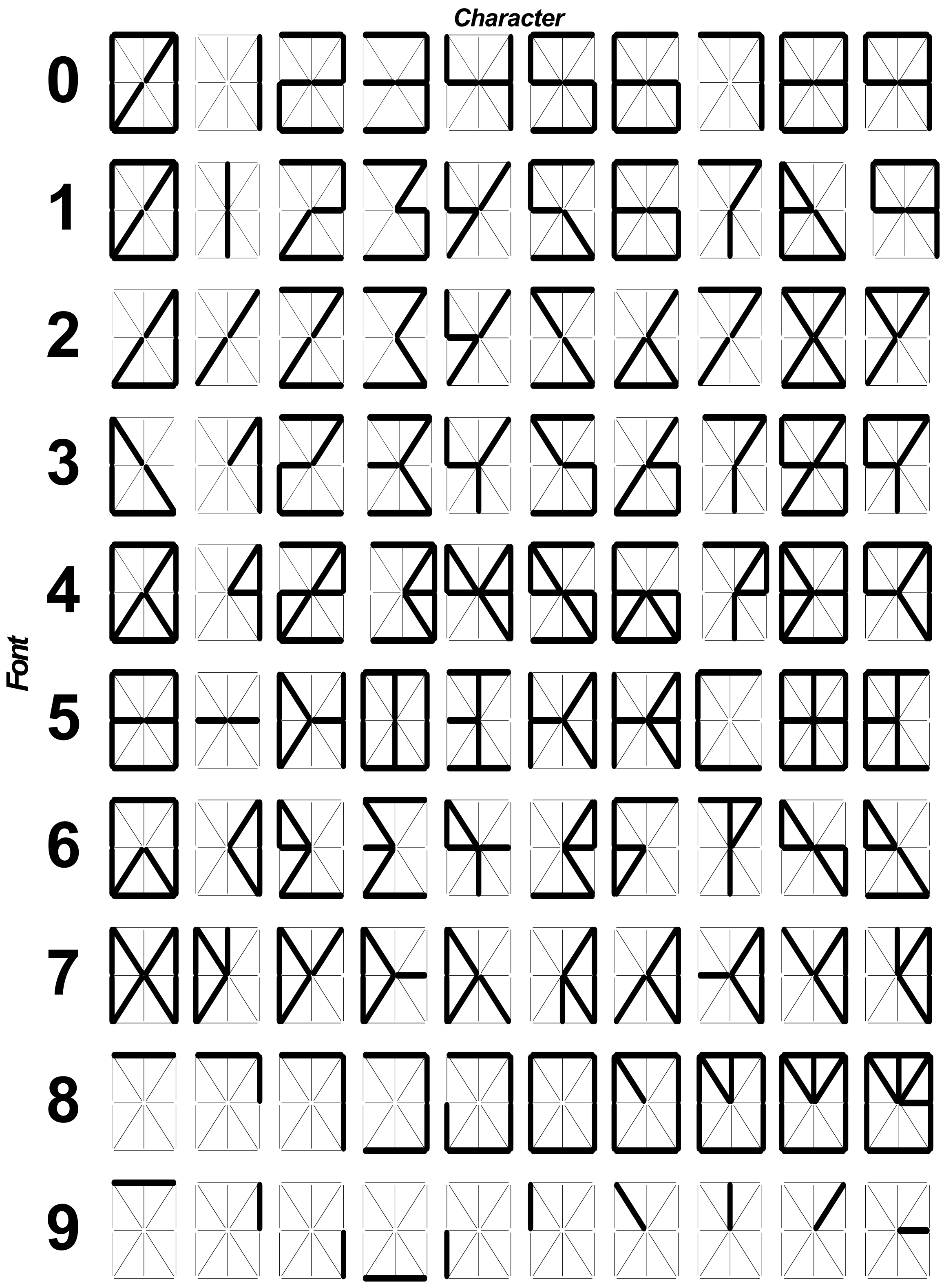
0 = Seven Segment – Standard Seven Segment Display Font
1 = Burroughs Fourteen Segment – Standard 14 Segment Display Font
2 = Leaning – Numbers an Italic Emphasis
3 = Formal – Digits Which Might Appear Somewhere Important
4 = Chamfer – A Font For Carpenters and Structural Engineers
5 = Secret – Lean Your Head Over To The Left
6 = Caitlin – My Daughter’s Hard Work, Proving Fontmaking is Childsplay
7 = Clockalogue – When Incremented Sequentially The Time Ticks By
8 = Linear Bargraph – The Number of Lights Lit Increases
9 = Nixie – Standard 0 to 9 Independent Cathode Drive
U = User Defined Characters

Alpha Characters: Fonts 0-9

Note that only upper case alpha characters are accepted for fonts 0-9, any lower case characters or characters not shown below will be ignored.



Numeric Characters: Fonts 0-9



[S]peed of effects command

```
$B7Sxxxxxx<cr>
```

Where xxxxx are the new effect speeds, position dependant. Valid Speed Number values are 0 to 9. This command structure works in the same way as $B7F and $B7E. A speed of "0" results in an immediate character change upon command receipt where "1" to "9" offers progressively slower transition speeds.

To change the effects speed of the second and fifth positions to fast and slow respectively and leave all other positions unchanged:

```
$B7S!0!!9!!!<cr>
```

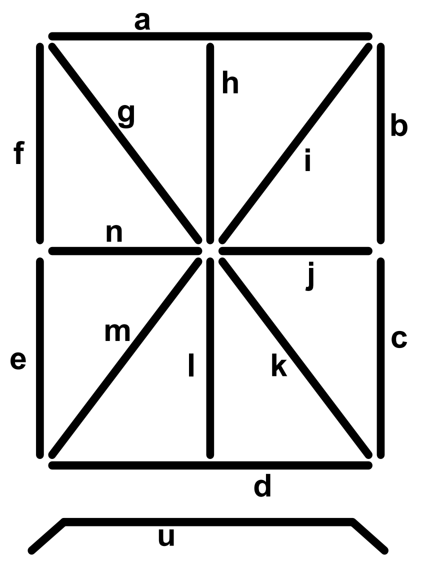
[W]rite a user defined character (UDC) to EEPROM

```
$B7Wcbbbbbbbbbbbbbbb<cr>
```

Where **c** is the ASCII alias character to be reprogrammed: ASCII 10 to 125 (<lf> to "}"), except <cr> and "!".

Where **bbbbbbbbbbbbbbb** is the 15 bit pattern used to define the new character.

The user defined characters are stored in an area of eeprom reserved specifically for the user so in no way can making changes to the UDC's affect program operation or the built in fonts, which makes dealing with UDC's a very safe practice. The use of eeprom memory means that the UDC's are retained after a power cycle. Each user character is assigned an ASCII alias which can then be easily recalled either directly from the keyboard of your terminal device, or by sending the alias character from the controlling device.



The segments of the B7971 and ZM1350 are denoted by letters as shown in the diagram. The group of letters **bbbbbbbbbbbbbbb** in the command refers to the segments in the following order: ***abcdefghijklmnu***. Once you have designed the new character, by entering the assignment of the segments into the corresponding position they occupy in the command, where a '1' denotes an on cathode and a '0' denotes an off cathode segment, the new character will be stored.

Example: Writing a UDC Bit Pattern to EEPROM

To produce a character which is made up from segments ***a,b,c,d*** and ***j*** and to store this new character in the location currently assigned to the character '**c**' (the alias character):

```
$B7Wc111100000100000<cr>
     abcdefghijklmnu
```

In order to make use of the new character include it into a display command string, referring to it by its original ascii definition, in this case ‘c’. It is important to note that the font which is assigned to the user defined characters does not have a number, but a single letter, the letter ’U’ (in uppercase). So for example, to display this new character on position 4 tube we first need to set the number 4 position to display the UDC:

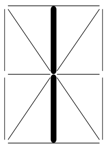
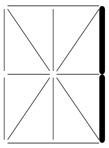
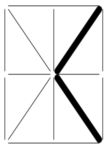
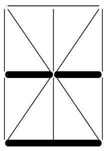
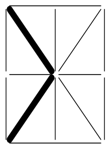
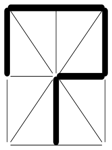
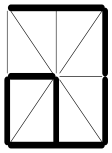
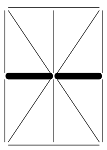
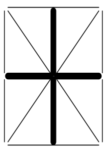
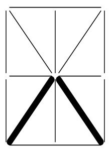
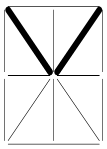
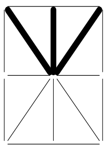
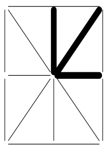
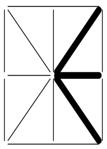
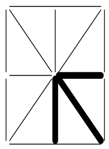
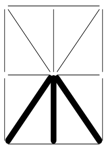
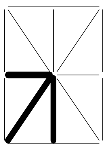
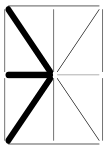
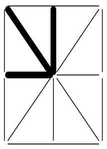
```
$B7F!!!U!!!!<cr>
```

And now to display it:

```
$B7M!!!c!!!!!<cr>
```

Initial UDC Settings

Some UDC Characters have already been programmed. These characters can be overwritten as required. Character codes 0 to 9 will display 7 segment right side justified numbers "0" to "9" which can be displayed under the [U]ser font but not modified.

: 58 	; 59 	< 60 	= 61 	> 62 
? 63 	@ 64 	A 65 	B 66 	C 67 
D 68 	E 69 	F 70 	G 71 	H 72 
I 73 	J 74 	K 75 	L 76 	x Else