MSX Article

Menu bar on Screen 0

Summary

This article describes a solution to create a menu bar on screen 0 who inverts the colors of the selected option, using the character pattern table.

1- Introduction

A computer program generally comes up with several utilities. Thus, the access to these resources may be a difficult task, depending on how the programmer establishes the way to achieve them. In the past, it was common to define keys combination to access resources, which was sometimes hard to memorize and discover them.

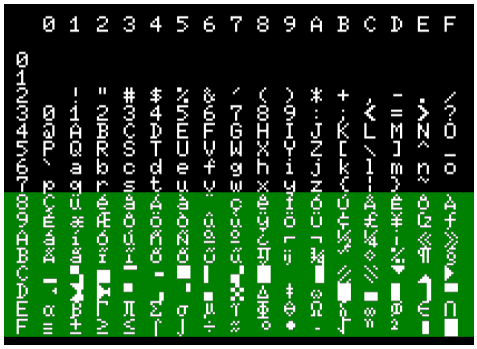
The menu bar is a solution the programmer has to offer his users as an easy way to discover and access resources available on his product. But before we start, we will take a look on MSX character table.

2- Characters on text mode

The MSX text-mode screens have two tables used for generating characters. The first one maps the characters that will be displayed on each screen position. This is the name table. The second table contains each character pattern, the charcter table. Each character has a code that identify it uniquely and they are used on both tables.

MSX follows an international character coding standard called ASCII. The ASCII defines the first 128 characters, ranging from 0 to 127. Thus, the character table uses 256 characters symbols. The values greater than 127 are generally destined to graphic symbols and special letters. This area is customized on each MSX.

Figure 2.1 shows two character tables from different MSXs. The black area is the standard ASCII and common for all MSXs. The green area is costumized. The Brazilian MSX (a) has latin characters on it, while the Turbo-R (b) has Japanese ideograms.



a) MSX Expert Gradiente 1.1

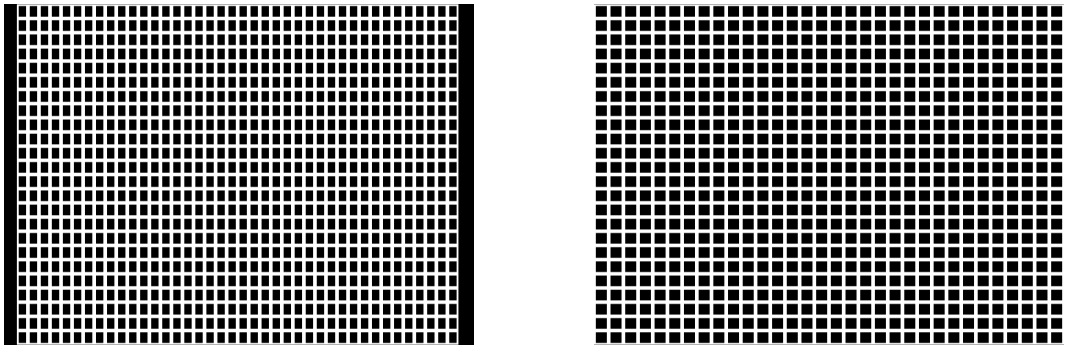
b) MSX Turbo-R GT

Figure 2.1. Example of different character tables on MSX.

On figure 2.1, the letter “M” is located at line 4 and column D, forming the hexadecimal number 4D. Converting this value to decimal, we conclude that the letter “M” has ASCII code equal to 77.

2.1. The Screen Division

The screen 0 divides the space into blocks of 8x6 pixels and the screen 1 divides into 8x8 pixels. Each block defines a character. According to that, each line on screen 0 has 40 characters, while the screen 1 has 32 characters. Both screens have 24 lines. See figure 2.2.



a) Screen 0 b) Screen 1

Figure 2.2. Text mode screens character division.

2.2. The Name Table

The name table maps each character block on screens 0 or 1 (figure 2.2), indicating which character will be displayed there. The blocks are numbered as follows:

Screen 0	Screen 1
0 1 2 3 … 39 40 41 42 43 … 79 … 920 921 922 923 … 959	0 1 2 3 … 31 32 33 34 35 … 63 … 736 737 738 739 … 767

Table 2.1. Blocks numeration.

For example, the top-left block on screen 0 is the name table entry number 0, while the top-right block on screen 1 is the entry #31. By the time we change the values on each entry, we are changing the displayed character, as seen on figure 2.3.

a) Screen 0 b) Screen 1

Figure 2.3. Example of name table value changed.

The name table on screen 0 starts at VRAM address 0, while on screen 1 starts at address 6144. The whole addressing:

	Initial address	Final address	Size
Screen 0	0	959	960 bytes
Screen 1	6144	6911	768 bytes

Table 2.2. Name table addresses.

The Basic instruction VPOKE modifies a VRAM address. In order to change the character displayed on a given block, we must do:

VPOKE initial_address + block_number, ascii_code

On the example before, we used the VPOKE 0,67 to draw the letter “A” on screen 0 top-left. For the screen 1 top-right, we used the VPOKE 6144+31,67 to draw the letter “A”.

In order to identify the block number, use the following equations:

$BN = X + (Y \times 40)$ Screen 0

$BN = X + (Y \times 32)$ Screen 1

Where *X* is the column, *Y* is the line and *BN* is the block number.

2.3. The Character Table

The character table defines each character pattern that will be displayed on screen. This table has 256 characters, and each code is used to refer the corresponding character pattern.

For both screens, each character pattern uses 8x8 monochromatic pixels and takes 8 bytes. So, the table has 256x8 or 2048 bytes size. The screen 0 only considers the first 6 columns on each character.

The character table addresses are shown on table 2.3.

	Initial address	Final address	Size
Screen 0	2048	4095	2048 bytes
Screen 1	0	2047	2048 bytes

Table 2.3. Character table addresses.

The starting address from each character is calculated as follows:

$P = 2048 + (C \times 8)$ Screen 0

$P = (C \times 8)$ Screen 1

Where *P* is the initial address and *C* is the ASCII code.

For example, the letter “A” has ASCII code 65. So, the initial address on VRAM for this letter is:

2048 + (65 x 8) = 2568

As told before, each character has 8x8 pixels. Each byte represents a line, where each bit represents a column. The value 0 indicates the background color (empty), while 1 indicates the foreground color (draw). This is the same pattern used on sprites.

The letter “A” has the following configuration:

```
VRAM   Bits
2568 = 0 0 1 0 0 0 0 0
2569 = 0 1 0 1 0 0 0 0
2570 = 1 0 0 0 1 0 0 0
2571 = 1 0 0 0 1 0 0 0
2572 = 1 1 1 1 1 0 0 0
2573 = 1 0 0 0 1 0 0 0
2574 = 1 0 0 0 1 0 0 0
2575 = 0 0 0 0 0 0 0 0
```

VPOKE 2568, &B00100000

The character pattern has always 8x8 pixels. Thus, screen 0 will only take 6x8 pixels.

The letter “A” pattern is defined on VRAM addresses from 2568 to 2575.

We can freely modify the character patterns. For that, we must change the 8 lines correponding to a character. It is also possible to save or load character tables:

BSAVE “FONT.ALF”, 2048, 4095, S
BSAVE “FONT.ALF”, 0, 2047, S

Obs: Once we are changing the pattern of the letter “A”, this modification is imediatelly reflected on all “A”s referenced by the name table.

The next program draws the pattern of a given charcter.

```
10 INPUT "ASCII code";A
20 E = 2048 + A*8
30 FOR I=E TO E+7
40 V$ = BIN$(VPEEK(I))
50 V$ = RIGHT$("0000000"+V$,8)
60 PRINT V$
70 NEXT I
```

3- Inverting the character's colors

The most menu bars inverts the colors of the current option text. For achieve that on screen 0, we have to invert the bits from each character on the character table. The NOT operator is used to invert the data. For example, let's see how to invert the colors of the letter “A” on the character table:

```
10 FOR E=2568 TO 2575
20 VPOKE E,&HFF AND (NOT VPEEK(E))
30 NEXT E
```

Once MSX operations returns numbers gerater than 1 byte, we use the logical operation *&HFF AND <value>* to grant a 1-byte bumber.

As told on section 2.3, if we change a character pattern it will be reflected on all chacters with the same ASCII code. After running the previous program, all the letters “A” will be inverted, as depicted on figure 3.1.

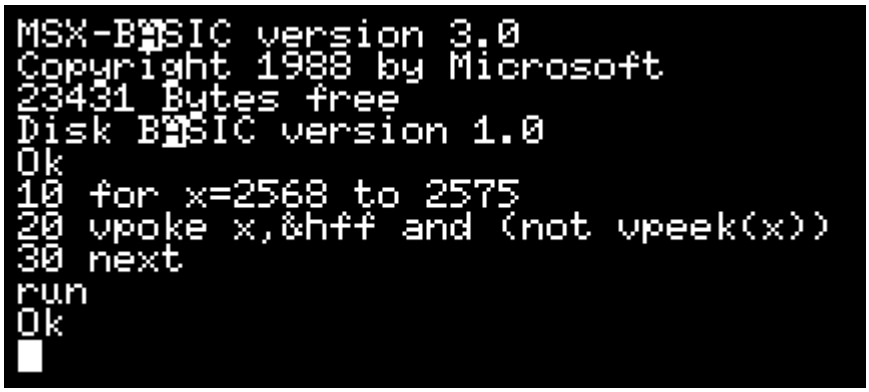


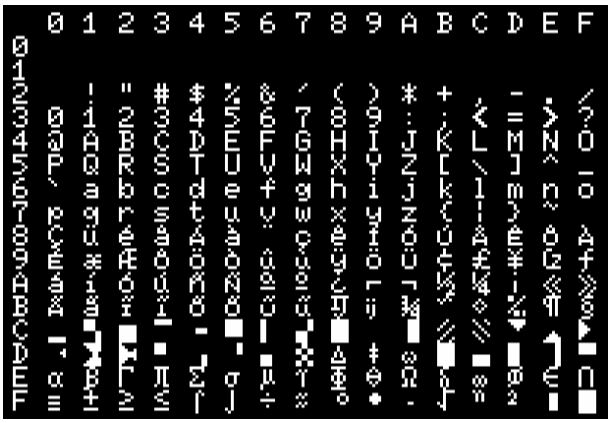
Figure 3.1. Inverting the colors of the character “A”.

4- Strategies to invert the colors on menu bar

Once a character pattern modification affects all characters having the same ASCII code on screen, it is necessary to elaborate a strategy that affect only the characters belonging to the menu bar. According to that, the first part of the character table cannot be modified.

Here, we suggest two strategies to achieve that goal: the first one is to copy the characters from the first part on the character table to the second one, inverting their colors. Thus, some special characters will be lost. The second one is to reserve *n* characters on an useless part of the character table, inverting the letters on real time. This would take only a number of characters corresponding to the menu bar's width.

Figure 4.1 presents the Expert MSX's (Brazil) characters, as well as the program in Basic to generate them.



```
10 SCREEN 0
20 FOR F=0 TO 15
30 LOCATE F*2+2,0
40 PRINT HEX$(F)
50 LOCATE 0,F+1
60 PRINT HEX$(F)
70 NEXT F
80 FOR X=0 TO 15
90 FOR Y=0 TO 15
100 LOCATE X*2+2,Y+1
110 A = X+(Y*16)
120 IF A < 32 THEN 140
130 PRINT CHR$(A)
140 NEXT Y,X
```

Figure 4.1 – Complete ASCII table from Gradiente Expert MSX 1.

To compose the ASCII code from a character on figure 4.1, we must concatenate the line number followed by the column number. For example, the letter “a” is located at line 6, column 1. So, the ASCII code in hexadecimal for “a” is 61 (97 in decimal).

Notice that from the ASCII code &H80 (128) to &HBF there are special latin characters. This area cannot be affected, if someone wants to use these characters on the menu.

Remembering the figure 3.1, imagine that we want to invert the top-left character on the screen, in that case, the letter “M”. Instead of inverting the “M” pattern on character table, we take another character from the table that will no be used by the menu, copy and invert the “M” pattern on that position. We can use, for instance, the ASCII code #250.

```
10 E1 = 2048 + 8*ASC("M") : E2 = 2048 + 8*250
20 FOR I=0 TO 7
30 VPOKE E2+I,&HFF AND (NOT VPEEK(E1+I))
40 NEXT I
```

The results can be seen on figure 4.2. Look at the inverted “M” located at the position &HFA (250).

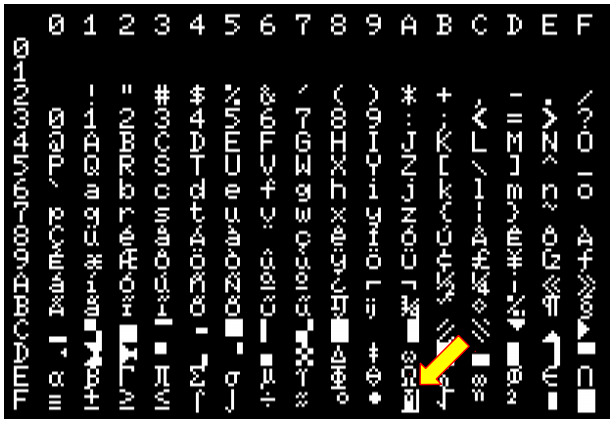


Figure 4.2. The character “M” inverted.

This procedure is not enough to produce the effect on the screen, once the ASCII code on the name table corresponding to the screen loation 0,0 is still 77, the original “M”. So, we must change the name table address 0 from 77 to 250. This procedure grants only the desired character on screen to be inverted.

This strategy can also be applied on *n* characters. Figure 4.3 shows a program and its results for creating a menu bar on the first line, using 8 characters. Notice that only that region corresponding to the bar was affected by the changes.

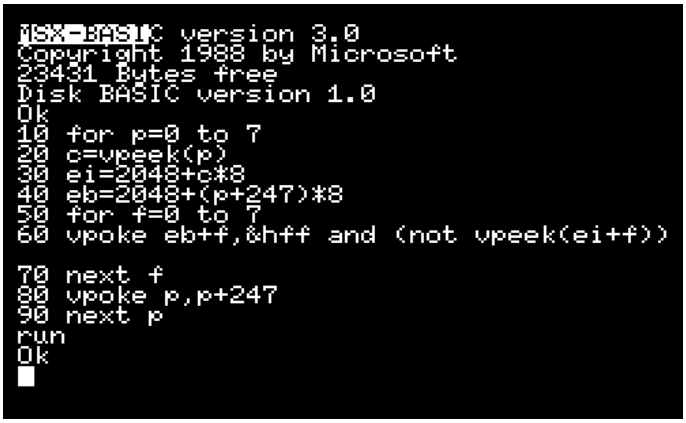


Figure 4.3. Colors inverted from a menubar with 8 charcters.

Obs: the instruction “SCREEN 0” resets all the tables and delete any modification.

5- Building a menubar

As seen before, both strategies perform modifications on the area of the name table corresponding to the current menu option. Once this bar is designed to navigate over the menu's options, it is necessary to preserve the ASCII codes under the bar in order to be restored every time the bar is moved.

The simpliest way to do that is to create an array and store the data there. So, every time we want to recover some item, we simply read that array.

The next program creates a simple menu using this strategy.

```
05 ' Draw menu
10 DIM OP$(5)
20 OP$(1) = "Edit    "
30 OP$(2) = "Cut     "
40 OP$(3) = "Load    "
50 OP$(4) = "Save    "
60 OP$(5) = "Exit    "
70 SCREEN 0:COLOR 15,0,0:WIDTH 40
80 PRINT"Menu":PRINT
90 FOR F=1 TO 5
100 PRINT OP$(F)
110 NEXT F
120 OP=1
130 GOSUB 500
200 ' Control
210 A$=INKEY$:IF A$="" THEN 210
220 A = ASC(A$)
230 IF A<>30 AND A<>31 THEN 210
240 IF A=30 AND OP=1 THEN 210
250 IF A=31 AND OP=5 THEN 210
260 LOCATE 0,OP+1:PRINT OP$(OP)
270 IF A$=CHR$(30) THEN OP=OP-1
280 IF A$=CHR$(31) THEN OP=OP+1
290 GOSUB 500
300 GOTO 210
500 ' Draw bar
510 E = (OP+1)*40
520 FOR P=0 TO 7
530 C = VPEEK(E+P)
540 EI = 2048 + C*8
550 EB = 2048 + (P+240)*8
560 FOR F=0 TO 7
570 VPOKE EB+F, &HFF AND (NOT VPEEK(EI+F))
580 NEXT F
590 VPOKE P+E,P+240
600 NEXT P
610 RETURN
```

The name table real time modification is quite slow for the MSX Basic.

The next program adopts the first strategy, where it “clones” and inverts the colors from the first part of the character table to the second. In addiction, it creats a second array list having the corresponding ASCII codes for the characters inverted.

```
05 ' Draw menu
06 SCREEN 0:COLOR 15,0,0:WIDTH 40
07 GOSUB 500
10 DIM OP$(5,2)
20 OP$(1,1) = "Edit    "
30 OP$(2,1) = "Cut     "
40 OP$(3,1) = "Load    "
50 OP$(4,1) = "Save    "
60 OP$(5,1) = "Exit    "
70 FOR F=1 TO 5
80 FOR C=1 TO 8
90 OP$(F,2) = OP$(F,2) + CHR$(ASC(MID$(OP$(F,1),C,1)) + 128)
100 NEXT C,F
110 PRINT"Menu":PRINT
120 FOR F=1 TO 5
130 PRINT OP$(F,1)
140 NEXT F
150 OP=1
160 GOTO 290
200 ' Control
210 A$=INKEY$:IF A$="" THEN 210
220 A = ASC(A$)
230 IF A<>30 AND A<>31 THEN 210
240 IF A=30 AND OP=1 THEN 210
250 IF A=31 AND OP=5 THEN 210
260 LOCATE 0,OP+1:PRINT OP$(OP,1)
270 IF A$=CHR$(30) THEN OP=OP-1
280 IF A$=CHR$(31) THEN OP=OP+1
290 LOCATE 0,OP+1:PRINT OP$(OP,2)
300 GOTO 210
500 ' Draw table
510 FOR C=32 TO 127
520 EI = 2048 + C*8
530 ED = 2048 + (C+128)*8
540 FOR F=0 TO 7
550 VPOKE ED+F, &HFF AND (NOT VPEEK(EI+F))
560 NEXT F,C
570 RETURN
```

The whole operation for inverting the characters takes a little bit of time, but the menu navigation is quite faster than the previous strategy.

The menu event codes are listed below.

```
if a=30 then <up>
if a=31 then <down>
if a=29 then <left>
if a=28 then <right>
if a=32 then <space>
if a=27 then <ESC>
if a=13 then <enter>
```

6 - Extra: MSX 1 VRAM map

<table>
<tr><th>VRAM Address</th><th>Screen 0</th><th>Screen 1</th><th>Screen 2</th><th>Screen 3</th></tr>
<tr><td>0</td><td rowspan="2">Name
0</td><td rowspan="3">Character
7</td><td rowspan="5">Character
12</td><td rowspan="3">Character
17</td></tr>
<tr><td>959</td></tr>
<tr><td>2947</td><td></td></tr>
<tr><td>2048</td><td rowspan="2">Character
2</td><td rowspan="3"></td><td rowspan="2">Name
15</td></tr>
<tr><td>4095</td></tr>
<tr><td>6143</td><td rowspan="9"></td><td rowspan="3"></td></tr>
<tr><td>6144</td><td rowspan="2">Name
5</td><td rowspan="2">Name
10</td></tr>
<tr><td>6911</td></tr>
<tr><td>6912</td><td rowspan="2">Sprite Attribute
8</td><td rowspan="2">Sprite Attribute
13</td><td rowspan="2">Sprite Attribute
18</td></tr>
<tr><td>7039</td></tr>
<tr><td>8192</td><td rowspan="3">Color
6</td><td rowspan="3">Color
11</td><td rowspan="3"></td></tr>
<tr><td>8223</td></tr>
<tr><td>14335</td></tr>
<tr><td>14336</td><td rowspan="2">Sprite Pattern
9</td><td rowspan="2">Sprite Pattern
14</td><td rowspan="2">Sprite Pattern
19</td></tr>
<tr><td>16383</td></tr>
</table>

Source: CPU-MSX magazine #15.

Obs: the numbers below the table names are the parameters passed to the Basic intruction BASE(n), which returns the initial address of each table.

7- Credits and references

This article was originally written in portuguese and translated into English by Marcelo Silveira.

Release: October 2003.
Revision 1: July 2017.
Revision 2: November 2017.

E-mail: flamar98@hotmail.com
Homepage: marmsx.msxall.com

References:

- MSX Red Book, Avalon Software, editora Mc Grall Hill.