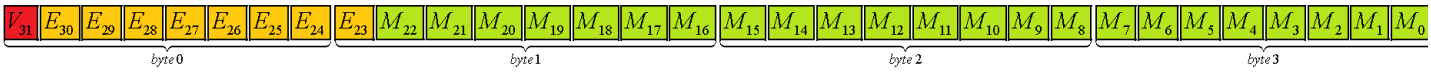
Real numbers inside a computer: float, double, long double and more



SIGN EXPONENT MANTISSA

To bias a transistor means to give him a positive or negative preload, *a biasing voltage*. An unbiased transistor is one without any preload. In that way **float** is using a biased exponent with preload_value: -127 for values between roughly 10^{-45} and 10^{38}.

$$x_{float} = \underbrace{s}_{\text{algebraic sign}} \cdot \underbrace{m}_{\text{mantissa (significand)}} \cdot e^{\text{exponend-bias}}$$

with algebraic sign: $s := (-1)^V \quad \Rightarrow \quad V = 0 \;\; then \;\; s = 1 \;\; and \;\; V = 1 \;\; then \;\; s = -1.$

The picture shows: $\underbrace{V_{31}}_{(-1)^{V_{31}}} \; \underbrace{E_{30}}_{E_{30}\cdot 2^7} \cdots \underbrace{E_{23}}_{E_{23}\cdot 2^0} \; \underbrace{M_{22}}_{M_{22}\cdot\frac{1}{2^1}} \cdots \underbrace{M_0}_{M_0\cdot\frac{1}{2^{23}}}$ which are 4 byte (32 bit). Again:

$$x_{float} = (-1)^{V_{31}} \cdot \left(1 + \frac{M_{22}}{2^1} + \frac{M_{21}}{2^2} + \frac{M_{20}}{2^3} + \cdots + \frac{M_0}{2^{23}}\right) \cdot 2^{\left(E_{30}\cdot 2^7 + \cdots + E_{23}\cdot 2^0\right) - 127}$$

Thereby the read “1” and “-127” are dark values – not stored inside the 4 byte (32 bit) of any float variable x.

Example: We ask a DIGILENT Max32_Pic32_board with a ET-LCD6610-NXP(Philips) display in which way

float x = 15.123456; is stoared.

```
#include <NXP_FONT.h>

#define myBL 2       // Digital 2 --> BL    pinMode(myBL, OUTPUT);
#define myCS 3       // Digital 3 --> #CS   pinMode(myCS, OUTPUT);
#define myCLK 4      // Digital 4 --> SCLK  pinMode(myCLK, OUTPUT);
#define mySDA 5      // Digital 5 --> SDATA pinMode(mySDA, OUTPUT);
#define myRESET 6    // Digital 6 --> #RESET pinMode(myRESET, OUTPUT);

//Philips(NXP):PCF8833 Header
#define NOP 0x00           // nop
#define SWRESET  0x01      // software reset
#define BSTROFF  0x02      // booster voltage OFF
#define BSTRON   0x03      // booster voltage ON
#define RDDIDIF  0x04      // read display identification
#define RDDST    0x09      // read display status
#define SLEEPIN  0x10      // sleep in
#define SLEEPOUT 0x11      // sleep out
#define PTLON    0x12      // partial display mode
#define NORON    0x13      // display normal mode
#define INVOFF   0x20      // inversion OFF
#define INVON    0x21      // inversion ON
#define DALO     0x22      // all pixel OFF
#define DAL      0x23      // all pixel ON
#define SETCON   0x25      // write contrast
#define DISPOFF  0x28      // display OFF
#define DISPON   0x29      // display ON
#define CASET    0x2A      // column address set
#define PASET    0x2B      // page address set
#define RAMWR    0x2C      // memory write
#define RGBSET   0x2D      // colour set
#define PTLAR    0x30      // partial area
#define VSCRDEF  0x33      // vertical scrolling definition
#define TEOFF    0x34      // test mode
#define TEON     0x35      // test mode
#define MADCTL   0x36      // memory access control
#define SEP      0x37      // vertical scrolling start address
#define IDMOFF   0x38      // idle mode OFF
```

The used font is inside #include <NXP_FONT.h>
C:\heute\juni_2011\Arduino\mpide\hardware\pic32\libraries\myLCD.

To pick up a byte of a float we use a makro:
#define byte_of_x(x, n) *((byte*)(&x) + n)

float x_float = 15.123456;

byte_of_x(x_float, 0) is $V_{31}E_{30}\cdots E_{24}$

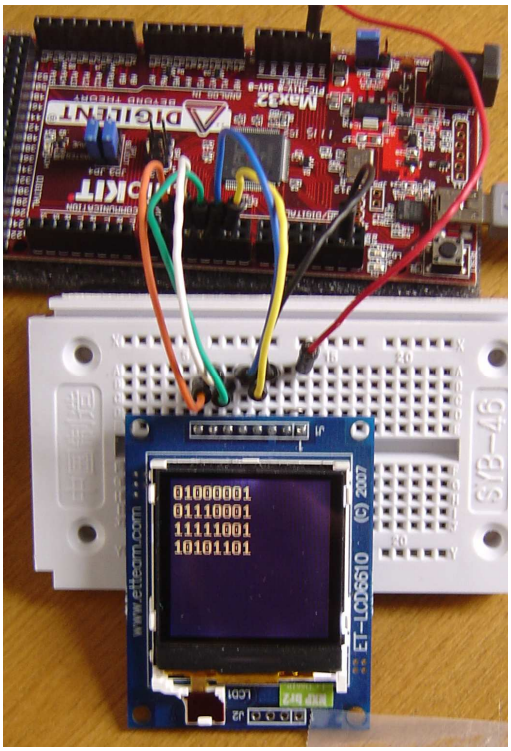
byte_of_x(x_float, 1) is $E_{23}M_{22}\cdots M_{16}$

byte_of_x(x_float, 2) is $M_{15}\cdots M_8$

byte_of_x(x_float, 3) is $M_7\cdots M_0$

```
#define IDMON    0x39     // idle mode ON
#define COLMOD   0x3A     // interface pixel format
#define SETVOP   0xB0     // set Vop
#define BRS      0xB4     // bottom row swap
#define TRS      0xB6     // top row swap
#define DISCTR   0xB9     // display control
//#define DAOR   0xBA     // data order(DOR)
#define TCDFE    0xBD     // enable/disable DF temperature compensation
#define TCVOPE   0xBF     // enable/disable Vop temp comp
#define EC       0xC0     // internal or external oscillator
#define SETMUL   0xC2     // set multiplication factor
#define TCVOPAB  0xC3     // set TCVOP slopes A and B
#define TCVOPCD  0xC4     // set TCVOP slopes c and d
#define TCDF     0xC5     // set divider frequency
#define DF8COLOR 0xC6     // set divider frequency 8-color mode
#define SETBS    0xC7     // set bias system
#define RDTEMP   0xC8     // temperature read back
#define NLI      0xC9     // n-line inversion
#define RDID1    0xDA     // read ID1
#define RDID2    0xDB     // read ID2
#define RDID3    0xDC     // read ID3

// Font sizes
#define SMALL 0
#define MEDIUM 1
#define LARGE 2

// Booleans
#define NOFILL 0
#define FILL 1

// 12-bit color definitions
#define WHITE 0xFFF
#define BLACK 0x000
#define RED 0xF00
#define GREEN 0x0F0
#define BLUE 0x00F
#define CYAN 0x0FF
#define MAGENTA 0xF0F
#define YELLOW 0xFF0
#define BROWN 0xB22
#define ORANGE 0xFA0
#define PINK 0xF6A

#define CS_0 digitalWrite(myCS, LOW);
#define CS_1 digitalWrite(myCS, HIGH);
#define CLK_0 digitalWrite(myCLK, LOW);
#define CLK_1 digitalWrite(myCLK, HIGH);
#define SDA_0 digitalWrite(mySDA, LOW);
#define SDA_1 digitalWrite(mySDA, HIGH);
#define RESET_0 digitalWrite(myRESET, LOW);
#define RESET_1 digitalWrite(myRESET, HIGH);
#define BL_0 digitalWrite(myBL, LOW);
#define BL_1 digitalWrite(myBL, HIGH);
/* End of Define Philips(NXP):PCF8833 Header */

PROGMEM const unsigned char FONT6x8[] = F_6x8;
PROGMEM const unsigned char FONT8x8[]  = F_8x8;
PROGMEM const unsigned char FONT8x16[] = F_8x16;

//----------------Function prototypes---------------------

void sendCMD(byte);
void sendData(byte);
void shiftBits(byte);
void lcd_init();
void draw_color_bar();
void lcd_clear(uint16_t, byte, byte, byte, byte);
void LCDPutStr(char*, int, int, int, int, int);
void LCDPutChar(char, int, int, int, int, int);
void LCDSetLine(int, int, int, int, int);
void LCDSetRect(int, int, int, int, unsigned char fill, int);
void LCDSetCircle(int, int, int, int);
void LCDSetPixel(byte, byte, int);
void LCDSetXY(byte, byte);

#define byte_of_x(x, n)  *((byte*)(&x) + n)

char *my_ftoa(float val, char *str)
{
 //static char buffer[10];
 char *cp; cp=str;
 int v, v0, rest, rest0;
 char c;
 v0 = (int)val; v=v0;
 rest0=(int)((val-(int)val)*10000000); rest = rest0;
 do {
   v /= 10;
   cp++;
 } while(v != 0);
 do {
   rest /= 10;
   cp++;
 } while(rest != 0);
 cp++; //wegen ','
 *cp-- = 0;
 do {
   c = rest0 % 10;
   rest0 /= 10;
   c += '0';
   *cp-- = c;
 } while(rest0 != 0);
 *cp-- = ',';
 do {
   c = v0 % 10;
   v0 /= 10;
   c += '0';
   *cp-- = c;
```

```
int i,j;
 for(i = sizeof(x_float) - 1; i >= 0; i--)
 {
  x_byte[i] = byte_of_x(x_float, i);
  txt[8]=0;
  for (j=0;j<8;j++){
   if (x_byte[i] & (1<<j))
     txt[7-j]='1';
   else
     txt[7-j]='0';
  }
  LCDPutStr(txt,50-15*i,2,LARGE,ORANGE,BLACK);
```



Bits of **float** x_float = 15.123456;

01000001 01110001 11111001 10101101

15,123456 = VEEEEEEE EMMMMMMM MMMMMMMM MMMMMMMM

$15/2=7$ *remainder* 1 *consider as least significant bit*

$7/2=3$ *remainder* 1

$3/2=1$ *remainder* 1

$1/2=0$ *remainder* 1 $\xrightarrow{\textit{in that way}}$ 0001111

```
 } while(v0 != 0);
 return  cp;
}

//------------Arduino-Main Code starts here--------------------

void setup()
{
 pinMode(myBL, OUTPUT);
 pinMode(myCS, OUTPUT);
 pinMode(myCLK, OUTPUT);
 pinMode(mySDA, OUTPUT);
 pinMode(myRESET, OUTPUT);
 digitalWrite(myBL, HIGH);
 digitalWrite(myCS, HIGH);
 digitalWrite(myCLK, HIGH);
 digitalWrite(mySDA, HIGH);
 digitalWrite(myRESET, HIGH);
 lcd_init();
 delay(500);
 /*bigin----------my_ftoa----------
 lcd_clear(BLACK,0,0,131,131);
 char  txt[16];
 #define pi 3.1234567
 my_ftoa(pi, txt);
 LCDPutStr(txt,50,2,LARGE,ORANGE,BLACK);
  /*---------------------------------------end*/

 lcd_clear(BLACK,0,0,131,131);
 char  txt[16];
 unsigned char x_byte[4];
 float x_float = 15.123456;
 int i,j;
 for(i = sizeof(x_float) - 1; i >= 0; i--)
 {
  x_byte[i] = byte_of_x(x_float, i);
  txt[8]=0;
  for (j=0;j<8;j++){
   if (x_byte[i] & (1<<j))
    txt[7-j]='1';
   else
    txt[7-j]='0';
  }
  LCDPutStr(txt,50-15*i,2,LARGE,ORANGE,BLACK);
 }

 /*
 LCDPutStr((char*)"LCD6610",          5, 40, LARGE,  YELLOW,  BLACK);
 LCDPutStr((char*)"132X132",          20, 40, LARGE,  CYAN,    BLACK);
 LCDPutStr((char*)"Color Graphic LCD", 37, 17, SMALL,  CYAN,    BLACK);
 LCDPutStr((char*)"ArduinoPIC_Max32",  50,  2, LARGE,  RED,     WHITE);
 LCDPutStr((char*)"SMALL GREEN",      70, 37, SMALL,  GREEN,   BLACK);
 LCDPutStr((char*)"MEDIUM BLUE",      81, 25, MEDIUM, BLUE,    BLACK);
 LCDPutStr((char*)"LARGE PINK",       90, 27, LARGE,  PINK,    BLACK);
 LCDPutStr((char*)"MEDIUM MAGENTA",   107, 12, MEDIUM, MAGENTA, BLACK);
 LCDPutStr((char*)"SMALL ORANGE",     119, 30, SMALL,  ORANGE,  BLACK);
 */
}

void loop()
{}

void sendCMD(byte data)
{
 CS_1
 CLK_0
 CS_0
 SDA_0
 CLK_1
 CLK_0

 shiftBits(data);
 CLK_0
 CS_1
}

void sendData(byte data) {

 CS_1
 CLK_0
 CS_0
 SDA_1
 CLK_1
 CLK_0

 shiftBits(data);
 CLK_0
 CS_1
}

void shiftBits(byte data)
{
 byte Bit;

 for (Bit = 0; Bit < 8; Bit++)    // 8 Bit Write
 {
  CLK_0       // Standby SCLK
  if((data&0x80)>>7)
  {
   SDA_1
  }
  else
  {
   SDA_0
  }
  CLK_1       // Strobe signal bit
  data <<= 1;  // Next bit data
 }
}
```

$$0,123456/\left(\frac{1}{2}\right)=0,123456\cdot 2=\underbrace{0,246912}_{\uparrow \textit{most significant bit}}$$

$$0,246912/\left(\frac{1}{2}\right)=0,246912\cdot 2=0,493824$$

$$0,493824/\left(\frac{1}{2}\right)=0,493824\cdot 2=0,987648$$

$$0,987648/\left(\frac{1}{2}\right)=0,987648\cdot 2=1,975296$$

$$0,975296/\left(\frac{1}{2}\right)=0,975296\cdot 2=1,950592$$

$$0,950592/\left(\frac{1}{2}\right)=0,950592\cdot 2=1,901184$$

$$0,901184/\left(\frac{1}{2}\right)=0,901184\cdot 2=1,802368$$

$$0,802368/\left(\frac{1}{2}\right)=0,802368\cdot 2=1,604736$$

$$0,604736/\left(\frac{1}{2}\right)=0,604736\cdot 2=1,209472$$

$$0,209472/\left(\frac{1}{2}\right)=0,209472\cdot 2=0,418944$$

$$0,418944\cdot 2=0,837888$$
$$0,837888\cdot 2=1,675776$$
$$0,675776\cdot 2=1,351552$$
$$0,351552\cdot 2=0,703104$$
$$0,703104\cdot 2=1,406208$$
$$0,406208\cdot 2=0,812416$$
$$0,812416\cdot 2=1,624832$$
$$0,624832\cdot 2=1,249664$$
$$0,249664\cdot 2=0,499328$$
$$0,499328\cdot 2=0,998656$$
$$0,998656\cdot 2=1,997312$$
$$0,997312\cdot 2=1,994624$$
$$0,994624\cdot 2=1,989248$$
$$0,989248\cdot 2=1,978496$$

$$15,123456=1111,0001111100110101101100111111$$

normalize to $=1,1110001111110011010110011111\cdot 2^{3}$; *comma shifted about* 3 *digits*

dark values $=1,1110001111110011010110011111\cdot 2^{3=130-127}$

Exponent : $130=10000010$; *algebraic sign : + is 0*

$$15,123456=01000001\ 01110001\ 11111001\ 10101100\ \underbrace{11111\cdots}_{\text{round to 1}}$$

$$x\_float=\underbrace{01000001}_{\text{byte 0}}\ \underbrace{01110001}_{\textit{byte 1}}\ \underbrace{11111001}_{\textit{byte 2}}\ \underbrace{10101101}_{\textit{byte 3}}$$ ↵ *we end as we started.*

```
void lcd_init()
{
  // Initial state
  CLK_0
  CS_1
  SDA_1

  // Hardware Reset LCD
  RESET_0
  delay(100);
  RESET_1
  delay(100);

  // Sleep out (commmand 0x11)
  sendCMD(SLEEPOUT);

  // Inversion on (command 0x20)
  //sendCMD(INVON);   // seems to be required for this controller
  sendCMD(INVOFF);

  // Color Interface Pixel Format (command 0x3A)
  sendCMD(COLMOD);
  sendData(0x03);   // 0x03 = 12 bits-per-pixel

  // Memory access controler (command 0x36)
  sendCMD(MADCTL);
  sendData(0xC8); // 0xC0 = mirror x and y, reverse rgb

  // Write contrast (command 0x25)
  sendCMD(SETCON);
  sendData(63); // contrast
  delay(1000);

  // Display On (command 0x29)
  sendCMD(DISPON);
}

void draw_color_bar()
{
  lcd_clear(RED,0,0,131,33);
  lcd_clear(GREEN,0,34,131,66);
  lcd_clear(BLUE,0,67,131,99);
  lcd_clear(WHITE,0,100,131,131);
}

void lcd_clear(uint16_t color, byte x0, byte y0, byte x1, byte y1)
{
  uint16_t xmin, xmax, ymin, ymax;
  uint16_t i;

  // best way to create a filled rectangle is to define a drawing box
  // and loop two pixels at a time
  // calculate the min and max for x and y directions
  xmin = (x0 <= x1) ? x0 : x1;
  xmax = (x0 > x1) ? x0 : x1;
  ymin = (y0 <= y1) ? y0 : y1;
  ymax = (y0 > y1) ? y0 : y1;

  // specify the controller drawing box according to those limits
  // Row address set (command 0x2B)
  sendCMD(PASET);
  sendData(xmin);
  sendData(xmax);

  // Column address set (command 0x2A)
  sendCMD(CASET);
  sendData(ymin);
  sendData(ymax);

  // WRITE MEMORY
  sendCMD(RAMWR);

  // loop on total number of pixels / 2
  for (i = 0; i < ((((xmax - xmin + 1) * (ymax - ymin + 1)) / 2) + 1); i++)
  {
    // use the color value to output three data bytes covering two pixels
    // For some reason, it has to send blue first then green and red
    sendData((color << 4) | ((color & 0xF0) >> 4));
    sendData(((color >> 4) & 0xF0) | (color & 0x0F));
    sendData((color & 0xF0) | (color >> 8));
  }
}

void LCDPutStr(char *pString, int x, int y, int Size, int fColor, int bColor)
{
  // loop until null-terminator is seen
  while (*pString != 0x00)
  {
    // draw the character
    LCDPutChar(*pString++, x, y, Size, fColor, bColor);

    // advance the y position
    if (Size == SMALL)
    y = y + 6;

    else if (Size == MEDIUM)
    y = y + 8;

    else
    y = y + 8;

    // bail out if y exceeds 131
    if (y > 131) break;
  }
}

void LCDPutChar(char c, int x, int y, int size, int fColor, int bColor)
{
```

```
  int i,j;
  unsigned int  nCols;
  unsigned int  nRows;
  unsigned int  nBytes;
  unsigned char PixelRow;
  unsigned char Mask;
  unsigned int  Word0;
  unsigned int  Word1;
  unsigned char *pFont;
  unsigned char *pChar;
  unsigned char *FontTable[] = {(unsigned char *)FONT6x8,
                                (unsigned char *)FONT8x8,
                                (unsigned char *)FONT8x16};

  // get pointer to the beginning of the selected font table
  pFont = (unsigned char *)FontTable[size];

  // get the nColumns, nRows and nBytes
  nCols = *pFont;
  nRows = *(pFont + 1);
  nBytes = *(pFont + 2);
  // get pointer to the last byte of the desired character
  pChar = pFont + (nBytes * (c - 0x1F));
  // Row address set (command 0x2B)
  sendCMD(PASET);
  sendData(x);
  sendData(x + nRows - 1);
  // Column address set (command 0x2A)
  sendCMD(CASET);
  sendData(y);
  sendData(y + nCols - 1);
  // WRITE MEMORY
  sendCMD(RAMWR);
  // loop on each row, working backwards from the bottom to the top
  for (i = nRows - 1; i >= 0; i--)
  {
    // copy pixel row from font table and then decrement row
    PixelRow = *pChar++;
    // loop on each pixel in the row (left to right)
    // Note: we do two pixels each loop
    Mask = 0x80;
    for (j = 0; j < nCols; j += 2)
          {
      // if pixel bit set, use foreground color; else use the background color
      // now get the pixel color for two successive pixels
      if ((PixelRow & Mask) == 0)
        Word0 = bColor;
      else
        Word0 = fColor;
      Mask = Mask >> 1;

      if ((PixelRow & Mask) == 0)
        Word1 = bColor;
      else
        Word1 = fColor;
      Mask = Mask >> 1;

      // use this information to output three data bytes
      // For some reason, it has to send blue first then green and red
      sendData((Word0 << 4) | ((Word0 & 0xF0) >> 4));
      sendData(((Word0 >> 4) & 0xF0) | (Word1 & 0x0F));
      sendData((Word1 & 0xF0) | (Word1 >> 8));
    }
  }

  // terminate the Write Memory command
  sendCMD(NOP);
}

void LCDSetLine(int x0, int y0, int x1, int y1, int color)
{
  int dy = y1 - y0;
  int dx = x1 - x0;
  int stepx, stepy;
  if (dy < 0) { dy = -dy; stepy = -1; } else { stepy = 1; }
  if (dx < 0) { dx = -dx; stepx = -1; } else { stepx = 1; }
  dy <<= 1; // dy is now 2*dy
  dx <<= 1; // dx is now 2*dx
  LCDSetPixel(x0, y0, color);
  if (dx > dy)
  {
    int fraction = dy - (dx >> 1); // same as 2*dy - dx
    while (x0 != x1)
    {
      if (fraction >= 0)
      {
        y0 += stepy;
        fraction -= dx; // same as fraction -= 2*dx
      }
      x0 += stepx;
      fraction += dy; // same as fraction -= 2*dy
      LCDSetPixel(x0, y0, color);
    }
  }
  else
  {
    int fraction = dx - (dy >> 1);
    while (y0 != y1)
          {
      if (fraction >= 0)
            {
        x0 += stepx;
        fraction -= dy;
      }
      y0 += stepy;
      fraction += dx;
      LCDSetPixel(x0, y0, color);
    }
  }
```

```
}

void LCDSetRect(int x0, int y0, int x1, int y1, unsigned char fill, int color)
{
  int xmin, xmax, ymin, ymax;
  int i;

  // check if the rectangle is to be filled
  if (fill == FILL)
  {
    // best way to create a filled rectangle is to define a drawing box
    // and loop two pixels at a time
    // calculate the min and max for x and y directions
    xmin = (x0 <= x1) ? x0 : x1;
    xmax = (x0 > x1) ? x0 : x1;
    ymin = (y0 <= y1) ? y0 : y1;
    ymax = (y0 > y1) ? y0 : y1;

    // specify the controller drawing box according to those limits
    // Row address set (command 0x2B)
    sendCMD(PASET);
    sendData(xmin);
    sendData(xmax);

    // Column address set (command 0x2A)
    sendCMD(CASET);
    sendData(ymin);
    sendData(ymax);

    // WRITE MEMORY
    sendCMD(RAMWR);

    // loop on total number of pixels / 2
    for (i = 0; i < ((((xmax - xmin + 1) * (ymax - ymin + 1)) / 2) + 1); i++)
    {
      // use the color value to output three data bytes covering two pixels
      // For some reason, it has to send blue first then green and red
      sendData((color << 4) | ((color & 0xF0) >> 4));
      sendData(((color >> 4) & 0xF0) | (color & 0x0F));
      sendData((color & 0xF0) | (color >> 8));
    }
  }
  else
  {
    // best way to draw un unfilled rectangle is to draw four lines
    LCDSetLine(x0, y0, x1, y0, color);
    LCDSetLine(x0, y1, x1, y1, color);
    LCDSetLine(x0, y0, x0, y1, color);
    LCDSetLine(x1, y0, x1, y1, color);
  }
}

void LCDSetCircle(int x0, int y0, int radius, int color)
{
  int f = 1 - radius;
  int ddF_x = 0;
  int ddF_y = -2 * radius;
  int x = 0;
  int y = radius;
  LCDSetPixel(x0, y0 + radius, color);
  LCDSetPixel(x0, y0 - radius, color);
  LCDSetPixel(x0 + radius, y0, color);
  LCDSetPixel(x0 - radius, y0, color);
  while (x < y)
  {
    if (f >= 0)
          {
      y--;
      ddF_y += 2;
      f += ddF_y;
    }
    x++;
    ddF_x += 2;
    f += ddF_x + 1;
    LCDSetPixel(x0 + x, y0 + y, color);
    LCDSetPixel(x0 - x, y0 + y, color);
    LCDSetPixel(x0 + x, y0 - y, color);
    LCDSetPixel(x0 - x, y0 - y, color);
    LCDSetPixel(x0 + y, y0 + x, color);
    LCDSetPixel(x0 - y, y0 + x, color);
    LCDSetPixel(x0 + y, y0 - x, color);
    LCDSetPixel(x0 - y, y0 - x, color);
  }
}

void LCDSetPixel(byte x, byte y, int color)
{
  LCDSetXY(x, y);
  sendCMD(RAMWR);
  // For some reason, it has to send blue first then green and red
  sendData((color << 4) | ((color & 0xF0) >> 4));
  sendData(((color >> 4) & 0xF0));
  sendCMD(NOP);
}

void LCDSetXY(byte x, byte y)
{
  // Row address set (command 0x2B)
  sendCMD(PASET);
  sendData(x);
  sendData(x);

  // Column address set (command 0x2A)
  sendCMD(CASET);
  sendData(y);
  sendData(y);
}
```

Content of C:\heute\juni_2011\Arduino\mpide\hardware\pic32\libraries\myLCD\ NXP_FONT.h

```
#define F_6x8
{0x06,0x08,0x08,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x20,0x20,0x20,0x20,0x20,0x00,0x20,0x00,0x50,0x50,0x50,0x00,0x00,0x00,0x00,0x00,0x50,0x50,0xF8,0x50,0xF8,0x50,0x50,0x00,0x20,0x78,0xA0,0x70,0x28,0xF0,0x20,0
x00,0xC0,0xC8,0x10,0x20,0x40,0x98,0x18,0x00,0x40,0xA0,0xA0,0x40,0xA8,0x90,0x68,0x00,0x30,0x30,0x20,0x40,0x00,0x00,0x00,0x00,0x10,0x20,0x40,0x40,0x40,0x20,0x10,0x00,0x40,0x20,0x10,0x10,0x10,0x20,0x40,0x00,0x00,0x20,0xA8,0x70,0x70,0xA8,0x
20,0x00,0x00,0x20,0x20,0xF8,0x20,0x20,0x00,0x00,0x00,0x00,0x00,0x00,0x30,0x30,0x20,0x40,0x00,0x00,0x00,0xF8,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x30,0x30,0x00,0x00,0x08,0x10,0x20,0x40,0x80,0x00,0x00,0x70,0x88,0x88,0xA8,0x88,0x88
,0x70,0x00,0x20,0x60,0x20,0x20,0x20,0x20,0x70,0x00,0x70,0x88,0x08,0x70,0x80,0x80,0xF8,0x00,0xF8,0x08,0x10,0x30,0x08,0x88,0x70,0x00,0x10,0x30,0x50,0x90,0xF8,0x10,0x10,0x00,0xF8,0x80,0xF0,0x08,0x08,0x88,0x70,0x00,0x38,0x40,0x80,0xF0,0x88,0x
88,0x70,0x00,0xF8,0x08,0x08,0x10,0x20,0x40,0x80,0x00,0x70,0x88,0x88,0x70,0x88,0x88,0x70,0x00,0x70,0x88,0x88,0x78,0x08,0x10,0xE0,0x00,0x00,0x00,0x20,0x00,0x20,0x00,0x00,0x00,0x00,0x00,0x20,0x00,0x20,0x20,0x40,0x00,0x08,0x10,0x20,0x40,0x20
,0x10,0x08,0x00,0x00,0x00,0xF8,0x00,0xF8,0x00,0x00,0x00,0x40,0x20,0x10,0x08,0x10,0x20,0x40,0x00,0x70,0x88,0x08,0x30,0x20,0x00,0x20,0x00,0x70,0x88,0xA8,0xB8,0xB0,0x80,0x78,0x00,0x20,0x50,0x88,0x88,0xF8,0x88,0x88,0x00,0xF0,0x88,0x88,0xF0,0x
88,0x88,0xF0,0x00,0x70,0x88,0x80,0x80,0x80,0x88,0x70,0x00,0xF0,0x88,0x88,0x88,0x88,0x88,0xF0,0x00,0xF8,0x80,0x80,0xF0,0x80,0x80,0xF8,0x00,0xF8,0x80,0x80,0xF0,0x80,0x80,0x80,0x00,0x78,0x88,0x80,0x80,0x98,0x88,0x78,0x00,0x88,0x88,0x88,0xF8,
0x88,0x88,0x88,0x00,0x70,0x20,0x20,0x20,0x20,0x20,0x70,0x00,0x38,0x10,0x10,0x10,0x10,0x90,0x60,0x00,0x88,0x90,0xA0,0xC0,0xA0,0x90,0x88,0x00,0x80,0x80,0x80,0x80,0x80,0x80,0xF8,0x00,0x88,0xD8,0xA8,0xA8,0xA8,0x88,0x88,0x00,0x88,0x88,0xC8,0
xA8,0x98,0x88,0x88,0x00,0x70,0x88,0x88,0x88,0x88,0x88,0x70,0x00,0xF0,0x88,0x88,0xF0,0x80,0x80,0x80,0x00,0x70,0x88,0x88,0x88,0xA8,0x90,0x68,0x00,0xF0,0x88,0x88,0xF0,0xA0,0x90,0x88,0x00,0x70,0x88,0x80,0x70,0x08,0x88,0x70,0x00,0xF8,0xA8,0x2
0,0x20,0x20,0x20,0x20,0x00,0x88,0x88,0x88,0x88,0x88,0x88,0x70,0x00,0x88,0x88,0x88,0x88,0x88,0x50,0x20,0x00,0x88,0x88,0x88,0xA8,0xA8,0xA8,0x50,0x00,0x88,0x88,0x50,0x20,0x50,0x88,0x88,0x00,0x88,0x88,0x50,0x20,0x20,0x20,0x20,0x00,0xF8,0x08,
0x10,0x70,0x40,0x80,0xF8,0x00,0x78,0x40,0x40,0x40,0x40,0x40,0x78,0x00,0x00,0x80,0x40,0x20,0x10,0x08,0x00,0x00,0x78,0x08,0x08,0x08,0x08,0x08,0x78,0x00,0x20,0x50,0x88,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0xF8,0x00,0x60,0x
60,0x20,0x10,0x00,0x00,0x00,0x00,0x00,0x00,0x60,0x10,0x70,0x90,0x78,0x00,0x80,0x80,0xB0,0xC8,0x88,0xC8,0xB0,0x00,0x00,0x00,0x70,0x88,0x80,0x88,0x70,0x00,0x08,0x08,0x68,0x98,0x88,0x98,0x68,0x00,0x00,0x00,0x70,0x88,0xF8,0x80,0x70,0x00,0x10
,0x28,0x20,0x70,0x20,0x20,0x20,0x00,0x00,0x00,0x70,0x98,0x98,0x68,0x08,0x70,0x80,0x80,0xB0,0xC8,0x88,0x88,0x88,0x00,0x20,0x00,0x60,0x20,0x20,0x20,0x70,0x00,0x10,0x00,0x10,0x10,0x10,0x90,0x60,0x00,0x80,0x80,0x90,0xA0,0xC0,0xA0,0x90,0x00,0
x60,0x20,0x20,0x20,0x20,0x20,0x70,0x00,0x00,0x00,0xD0,0xA8,0xA8,0xA8,0xA8,0x00,0x00,0x00,0xB0,0xC8,0x88,0x88,0x88,0x00,0x00,0x00,0x70,0x88,0x88,0x88,0x70,0x00,0x00,0x00,0xB0,0xC8,0xC8,0xB0,0x80,0x80,0x00,0x00,0x68,0x98,0x98,0x68,0x08,0x
08,0x00,0x00,0xB0,0xC8,0x80,0x80,0x80,0x00,0x00,0x00,0x78,0x80,0x70,0x08,0xF0,0x00,0x20,0x20,0xF8,0x20,0x20,0x28,0x10,0x00,0x00,0x00,0x88,0x88,0x88,0x98,0x68,0x00,0x00,0x00,0x88,0x88,0x88,0x50,0x20,0x00,0x00,0x00,0x88,0x88,0xA8,0xA8,0x5
0,0x00,0x00,0x00,0x88,0x50,0x20,0x50,0x88,0x00,0x00,0x00,0x88,0x88,0x78,0x08,0x88,0x70,0x00,0x00,0xF8,0x10,0x20,0x40,0xF8,0x00,0x10,0x20,0x20,0x40,0x20,0x20,0x10,0x00,0x20,0x20,0x20,0x00,0x20,0x20,0x20,0x00,0x40,0x20,0x20,0x10,0x20,0x20,
0x40,0x00,0x40,0xA8,0x10,0x00,0x00,0x00,0x00,0x00,0x70,0xD8,0xD8,0x70,0x00,0x00,0x00,0x00}

#define F_8x8
{0x08,0x08,0x08,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x30,0x78,0x78,0x30,0x30,0x00,0x30,0x00,0x6C,0x6C,0x6C,0x00,0x00,0x00,0x00,0x00,0x6C,0x6C,0xFE,0x6C,0xFE,0x6C,0x6C,0x00,0x18,0x3E,0x60,0x3C,0x06,0x7C,0x18,0
x00,0x00,0x63,0x66,0x0C,0x18,0x33,0x63,0x00,0x1C,0x36,0x1C,0x3B,0x6E,0x66,0x3B,0x00,0x30,0x30,0x60,0x00,0x00,0x00,0x00,0x00,0x0C,0x18,0x30,0x30,0x30,0x18,0x0C,0x00,0x30,0x18,0x0C,0x0C,0x0C,0x18,0x30,0x00,0x00,0x66,0x3C,0xFF,0x3C,0x66,0x
00,0x00,0x00,0x30,0x30,0xFC,0x30,0x30,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x18,0x18,0x30,0x00,0x00,0x00,0x7E,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x18,0x18,0x00,0x03,0x06,0x0C,0x18,0x30,0x60,0x40,0x00,0x3E,0x63,0x63,0x6B,0x63,0x63
,0x3E,0x00,0x18,0x38,0x58,0x18,0x18,0x18,0x7E,0x00,0x3C,0x66,0x06,0x1C,0x30,0x66,0x7E,0x00,0x3C,0x66,0x06,0x1C,0x06,0x66,0x3C,0x00,0x0E,0x1E,0x36,0x66,0x7F,0x06,0x0F,0x00,0x7E,0x60,0x7C,0x06,0x06,0x66,0x3C,0x00,0x1C,0x30,0x60,0x7C,0x66,0
x66,0x3C,0x00,0x7E,0x66,0x06,0x0C,0x18,0x18,0x18,0x00,0x3C,0x66,0x66,0x3C,0x66,0x66,0x3C,0x00,0x3C,0x66,0x66,0x3E,0x06,0x0C,0x38,0x00,0x00,0x18,0x18,0x00,0x00,0x18,0x18,0x00,0x00,0x18,0x18,0x00,0x00,0x18,0x18,0x30,0x0C,0x18,0x30,0x60,0x
30,0x18,0x0C,0x00,0x00,0x00,0x7E,0x00,0x00,0x7E,0x00,0x00,0x30,0x18,0x0C,0x06,0x0C,0x18,0x30,0x00,0x3C,0x66,0x06,0x0C,0x18,0x00,0x18,0x00,0x3E,0x63,0x6F,0x69,0x6F,0x60,0x3E,0x00,0x18,0x3C,0x66,0x66,0x7E,0x66,0x66,0x00,0x7E,0x33,0x33,0x3E
,0x33,0x33,0x7E,0x00,0x1E,0x33,0x60,0x60,0x60,0x33,0x1E,0x00,0x7C,0x36,0x33,0x33,0x33,0x36,0x7C,0x00,0x7F,0x31,0x34,0x3C,0x34,0x31,0x7F,0x00,0x7F,0x31,0x34,0x3C,0x34,0x30,0x78,0x00,0x1E,0x33,0x60,0x60,0x67,0x33,0x1F,0x00,0x66,0x66,0x66,0x
7E,0x66,0x66,0x66,0x00,0x3C,0x18,0x18,0x18,0x18,0x18,0x3C,0x00,0x0F,0x06,0x06,0x06,0x66,0x66,0x3C,0x00,0x73,0x33,0x36,0x3C,0x36,0x33,0x73,0x00,0x78,0x30,0x30,0x30,0x31,0x33,0x7F,0x00,0x63,0x77,0x7F,0x7F,0x6B,0x63,0x63,0x00,0x63,0x73,0x7B
,0x6F,0x67,0x63,0x63,0x00,0x3E,0x63,0x63,0x63,0x63,0x63,0x3E,0x00,0x7E,0x33,0x33,0x3E,0x30,0x30,0x78,0x00,0x3C,0x66,0x66,0x66,0x6E,0x3C,0x0E,0x00,0x7E,0x33,0x33,0x3E,0x36,0x33,0x73,0x00,0x3C,0x66,0x30,0x18,0x0C,0x66,0x3C,0x00,0x7E,0x5A,0
x18,0x18,0x18,0x18,0x3C,0x00,0x66,0x66,0x66,0x66,0x66,0x66,0x7E,0x00,0x66,0x66,0x66,0x66,0x66,0x3C,0x18,0x00,0x63,0x63,0x63,0x6B,0x7F,0x77,0x63,0x00,0x63,0x63,0x36,0x1C,0x1C,0x36,0x63,0x00,0x66,0x66,0x66,0x3C,0x18,0x18,0x3C,0x00,0x7F,0x6
3,0x46,0x0C,0x19,0x33,0x7F,0x00,0x3C,0x30,0x30,0x30,0x30,0x30,0x3C,0x00,0x60,0x30,0x18,0x0C,0x06,0x03,0x01,0x00,0x3C,0x0C,0x0C,0x0C,0x0C,0x0C,0x3C,0x00,0x08,0x1C,0x36,0x63,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0xFF,0x18,
0x18,0x0C,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x3C,0x06,0x3E,0x66,0x3B,0x00,0x70,0x30,0x3E,0x33,0x33,0x33,0x6E,0x00,0x00,0x00,0x3C,0x66,0x60,0x66,0x3C,0x00,0x0E,0x06,0x3E,0x66,0x66,0x66,0x3B,0x00,0x00,0x00,0x3C,0x66,0x7E,0x60,0x3C,0x00,0x
1C,0x36,0x30,0x78,0x30,0x30,0x78,0x00,0x00,0x00,0x3B,0x66,0x66,0x3E,0x06,0x7C,0x70,0x30,0x36,0x3B,0x33,0x33,0x73,0x00,0x18,0x00,0x38,0x18,0x18,0x18,0x3C,0x00,0x06,0x00,0x06,0x06,0x06,0x66,0x66,0x3C,0x70,0x30,0x33,0x36,0x3C,0x36,0x73,0x0
0,0x38,0x18,0x18,0x18,0x18,0x18,0x3C,0x00,0x00,0x00,0x66,0x7F,0x7F,0x6B,0x63,0x00,0x00,0x00,0x7C,0x66,0x66,0x66,0x66,0x00,0x00,0x00,0x3C,0x66,0x66,0x66,0x3C,0x00,0x00,0x00,0x6E,0x33,0x33,0x3E,0x30,0x78,0x00,0x00,0x3B,0x66,0x66,0x3E,0x06,
0x0F,0x00,0x00,0x6E,0x3B,0x33,0x30,0x78,0x00,0x00,0x00,0x3E,0x60,0x3C,0x06,0x7C,0x00,0x08,0x18,0x3E,0x18,0x18,0x1A,0x0C,0x00,0x00,0x00,0x66,0x66,0x66,0x66,0x3B,0x00,0x00,0x00,0x66,0x66,0x66,0x3C,0x18,0x00,0x00,0x00,0x63,0x6B,0x7F,0x7F,0x
36,0x00,0x00,0x00,0x63,0x36,0x1C,0x36,0x63,0x00,0x00,0x00,0x66,0x66,0x66,0x3E,0x06,0x7C,0x00,0x00,0x7E,0x4C,0x18,0x32,0x7E,0x00,0x0E,0x18,0x18,0x70,0x18,0x18,0x0E,0x00,0x0C,0x0C,0x0C,0x00,0x0C,0x0C,0x0C,0x00,0x70,0x18,0x18,0x0E,0x18,0x1
8,0x70,0x00,0x3B,0x6E,0x00,0x00,0x00,0x00,0x00,0x00,0x1C,0x36,0x36,0x1C,0x00,0x00,0x00,0x00}

#define F_8x16
{0x08,0x10,0x10,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x18,0x3C,0x3C,0x3C,0x18,0x18,0x18,0x00,0x18,0x18,0x00,0x00,0x00,0
x00,0x00,0x63,0x63,0x63,0x22,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x36,0x36,0x7F,0x36,0x36,0x36,0x7F,0x36,0x36,0x00,0x00,0x00,0x00,0x0C,0x0C,0x3E,0x63,0x61,0x60,0x3E,0x03,0x03,0x43,0x63,0x3E,0x0C,0x0C,0x0
0,0x00,0x00,0x00,0x00,0x00,0x00,0x61,0x63,0x06,0x0C,0x18,0x33,0x63,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x1C,0x36,0x36,0x1C,0x3B,0x6E,0x66,0x66,0x3B,0x00,0x00,0x00,0x00,0x00,0x30,0x30,0x30,0x60,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,
0x00,0x00,0x00,0x00,0x0C,0x18,0x18,0x30,0x30,0x30,0x30,0x18,0x18,0x0C,0x00,0x00,0x00,0x00,0x00,0x00,0x18,0x0C,0x0C,0x06,0x06,0x06,0x06,0x0C,0x0C,0x18,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x42,0x66,0x3C,0xFF,0x3C,0x66,0x42,0x00,0x00,0x
00,0x00,0x00,0x00,0x00,0x00,0x00,0x18,0x18,0x18,0xFF,0x18,0x18,0x18,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x18,0x18,0x18,0x30,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0xFF,0x00,0x00,0x00,0x00,0x00,
0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x18,0x18,0x00,0x00,0x00,0x00,0x00,0x00,0x01,0x03,0x07,0x0E,0x1C,0x38,0x70,0xE0,0xC0,0x80,0x00,0x00,0x00,0x00,0x00,0x00,0x3E,0x63,0x63,0x63,0x6B,0x6B,0x63,0x63,0x63,0x3E,0x
00,0x00,0x00,0x00,0x00,0x00,0x0C,0x1C,0x3C,0x0C,0x0C,0x0C,0x0C,0x0C,0x0C,0x3F,0x00,0x00,0x00,0x00,0x00,0x00,0x3E,0x63,0x03,0x06,0x0C,0x18,0x30,0x61,0x63,0x7F,0x00,0x00,0x00,0x00,0x00,0x00,0x3E,0x63,0x03,0x03,0x1E,0x03,0x03,0x03,0x63,0x3E
,0x00,0x00,0x00,0x00,0x00,0x00,0x06,0x0E,0x1E,0x36,0x66,0x66,0x7F,0x06,0x06,0x0F,0x00,0x00,0x00,0x00,0x00,0x00,0x7F,0x60,0x60,0x60,0x7E,0x03,0x03,0x63,0x73,0x3E,0x00,0x00,0x00,0x00,0x00,0x00,0x1C,0x30,0x60,0x60,0x7E,0x63,0x63,0x63,0x63,0x
3E,0x00,0x00,0x00,0x00,0x00,0x00,0x7F,0x63,0x03,0x06,0x06,0x0C,0x0C,0x18,0x18,0x18,0x00,0x00,0x00,0x00,0x00,0x00,0x3E,0x63,0x63,0x63,0x3E,0x63,0x63,0x63,0x63,0x3E,0x00,0x00,0x00,0x00,0x00,0x00,0x3E,0x63,0x63,0x63,0x63,0x3F,0x03,0x03,0x06
,0x3C,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x18,0x18,0x00,0x00,0x00,0x18,0x18,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x18,0x18,0x00,0x00,0x00,0x18,0x18,0x18,0x30,0x00,0x00,0x00,0x00,0x00,0x06,0x0C,0x18,0x30,0x60,0x30,0x18,0
x0C,0x06,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x7E,0x00,0x00,0x7E,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x60,0x30,0x18,0x0C,0x06,0x0C,0x18,0x30,0x60,0x00,0x00,0x00,0x00,0x00,0x00,0x3E,0x63,0x63,0x06,0x0C,0x0C,0x0C,0x0
0,0x0C,0x0C,0x00,0x00,0x00,0x00,0x00,0x00,0x3E,0x63,0x63,0x6F,0x6B,0x6B,0x6E,0x60,0x60,0x3E,0x00,0x00,0x00,0x00,0x00,0x00,0x08,0x1C,0x36,0x63,0x63,0x63,0x7F,0x63,0x63,0x63,0x00,0x00,0x00,0x00,0x00,0x00,0x7E,0x33,0x33,0x33,0x3E,0x33,0x33,
0x33,0x33,0x7E,0x00,0x00,0x00,0x00,0x00,0x00,0x1E,0x33,0x61,0x60,0x60,0x60,0x60,0x61,0x33,0x1E,0x00,0x00,0x00,0x00,0x00,0x00,0x7C,0x36,0x33,0x33,0x33,0x33,0x33,0x33,0x36,0x7C,0x00,0x00,0x00,0x00,0x00,0x00,0x7F,0x33,0x31,0x34,0x3C,0x34,0x
30,0x31,0x33,0x7F,0x00,0x00,0x00,0x00,0x00,0x00,0x7F,0x33,0x31,0x34,0x3C,0x34,0x30,0x30,0x30,0x78,0x00,0x00,0x00,0x00,0x00,0x00,0x1E,0x33,0x61,0x60,0x60,0x6F,0x63,0x63,0x37,0x1D,0x00,0x00,0x00,0x00,0x00,0x00,0x63,0x63,0x63,0x63,0x7F,0x63
,0x63,0x63,0x63,0x63,0x00,0x00,0x00,0x00,0x00,0x00,0x3C,0x18,0x18,0x18,0x18,0x18,0x18,0x18,0x18,0x3C,0x00,0x00,0x00,0x00,0x00,0x00,0x0F,0x06,0x06,0x06,0x06,0x06,0x06,0x66,0x66,0x3C,0x00,0x00,0x00,0x00,0x00,0x00,0x73,0x33,0x36,0x36,0x3C,0
x36,0x36,0x33,0x33,0x73,0x00,0x00,0x00,0x00,0x00,0x00,0x78,0x30,0x30,0x30,0x30,0x30,0x30,0x31,0x33,0x7F,0x00,0x00,0x00,0x00,0x00,0x00,0x63,0x77,0x7F,0x6B,0x63,0x63,0x63,0x63,0x63,0x63,0x00,0x00,0x00,0x00,0x00,0x00,0x63,0x63,0x73,0x7B,0x7
F,0x6F,0x67,0x63,0x63,0x63,0x00,0x00,0x00,0x00,0x00,0x00,0x1C,0x36,0x63,0x63,0x63,0x63,0x63,0x63,0x36,0x1C,0x00,0x00,0x00,0x00,0x00,0x00,0x7E,0x33,0x33,0x33,0x3E,0x30,0x30,0x30,0x30,0x78,0x00,0x00,0x00,0x00,0x00,0x00,0x3E,0x63,0x63,0x63,
0x63,0x63,0x63,0x6B,0x6F,0x3E,0x06,0x07,0x00,0x00,0x00,0x00,0x7E,0x33,0x33,0x33,0x3E,0x36,0x36,0x33,0x33,0x73,0x00,0x00,0x00,0x00,0x00,0x00,0x3E,0x63,0x63,0x30,0x1C,0x06,0x03,0x63,0x63,0x3E,0x00,0x00,0x00,0x00,0x00,0x00,0xFF,0xDB,0x99,0x
18,0x18,0x18,0x18,0x18,0x18,0x3C,0x00,0x00,0x00,0x00,0x00,0x00,0x63,0x63,0x63,0x63,0x63,0x63,0x63,0x63,0x63,0x3E,0x00,0x00,0x00,0x00,0x00,0x00,0x63,0x63,0x63,0x63,0x63,0x63,0x63,0x36,0x1C,0x08,0x00,0x00,0x00,0x00,0x00,0x00,0x63,0x63,0x63
,0x63,0x63,0x6B,0x6B,0x7F,0x36,0x36,0x00,0x00,0x00,0x00,0x00,0x00,0xC3,0xC3,0x66,0x3C,0x18,0x18,0x3C,0x66,0xC3,0xC3,0x00,0x00,0x00,0x00,0x00,0x00,0xC3,0xC3,0xC3,0x66,0x3C,0x18,0x18,0x18,0x18,0x3C,0x00,0x00,0x00,0x00,0x00,0x00,0x7F,0x63,0
x43,0x06,0x0C,0x18,0x30,0x61,0x63,0x7F,0x00,0x00,0x00,0x00,0x00,0x00,0x3C,0x30,0x30,0x30,0x30,0x30,0x30,0x30,0x30,0x3C,0x00,0x00,0x00,0x00,0x00,0x00,0x80,0xC0,0xE0,0x70,0x38,0x1C,0x0E,0x07,0x03,0x01,0x00,0x00,0x00,0x00,0x00,0x00,0x3C,0x0
C,0x0C,0x0C,0x0C,0x0C,0x0C,0x0C,0x0C,0x3C,0x00,0x00,0x00,0x00,0x08,0x1C,0x36,0x63,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0xFF,0x00,0x00,0x00,0x18,0x18,0x0C,
0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x3C,0x46,0x06,0x3E,0x66,0x66,0x3B,0x00,0x00,0x00,0x00,0x00,0x00,0x70,0x30,0x30,0x3C,0x36,0x33,0x33,0x33,0x33,0x6E,0x00,0x00,0x00,0x00,0x00,0x00,0x
00,0x00,0x00,0x3E,0x63,0x60,0x60,0x60,0x63,0x3E,0x00,0x00,0x00,0x00,0x00,0x00,0x0E,0x06,0x06,0x1E,0x36,0x66,0x66,0x66,0x66,0x3B,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x3E,0x63,0x63,0x7E,0x60,0x63,0x3E,0x00,0x00,0x00,0x00,0x00,0x00
,0x1C,0x36,0x32,0x30,0x7C,0x30,0x30,0x30,0x30,0x78,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x3B,0x66,0x66,0x66,0x66,0x3E,0x06,0x66,0x3C,0x00,0x00,0x00,0x00,0x70,0x30,0x30,0x36,0x3B,0x33,0x33,0x33,0x33,0x73,0x00,0x00,0x00,0x00,0x00,0
x00,0x0C,0x0C,0x00,0x1C,0x0C,0x0C,0x0C,0x0C,0x0C,0x1E,0x00,0x00,0x00,0x00,0x00,0x00,0x06,0x06,0x00,0x0E,0x06,0x06,0x06,0x06,0x06,0x66,0x66,0x3C,0x00,0x00,0x00,0x00,0x70,0x30,0x30,0x33,0x33,0x36,0x3C,0x36,0x33,0x73,0x00,0x00,0x00,0x00,0x
00,0x00,0x1C,0x0C,0x0C,0x0C,0x0C,0x0C,0x0C,0x0C,0x0C,0x1E,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x6E,0x7F,0x6B,0x6B,0x6B,0x6B,0x6B,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x6E,0x33,0x33,0x33,0x33,0x33,0x33,0x00,0x00,0x00,0x0
0,0x00,0x00,0x00,0x00,0x00,0x3E,0x63,0x63,0x63,0x63,0x63,0x3E,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x6E,0x33,0x33,0x33,0x33,0x3E,0x30,0x30,0x78,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x3B,0x66,0x66,0x66,0x66,0x3E,0x06,0x06,0x0F,0x00,
0x00,0x00,0x00,0x00,0x00,0x00,0x6E,0x3B,0x33,0x30,0x30,0x30,0x78,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x3E,0x63,0x38,0x0E,0x03,0x63,0x3E,0x00,0x00,0x00,0x00,0x00,0x00,0x08,0x18,0x18,0x7E,0x18,0x18,0x18,0x18,0x1B,0x0E,0x00,0x00,0x
00,0x00,0x00,0x00,0x00,0x00,0x00,0x66,0x66,0x66,0x66,0x66,0x66,0x3B,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x63,0x63,0x36,0x36,0x1C,0x1C,0x08,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x63,0x63,0x63,0x6B,0x6B,0x7F,0x36,0x00,0x0
0,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x63,0x36,0x1C,0x1C,0x1C,0x36,0x63,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x63,0x63,0x63,0x63,0x63,0x3F,0x03,0x06,0x3C,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x7F,0x66,0x0C,0x18,0x30,0x63,0x7F,0x00,
0x00,0x00,0x00,0x00,0x00,0x0E,0x18,0x18,0x18,0x70,0x18,0x18,0x18,0x18,0x0E,0x00,0x00,0x00,0x00,0x00,0x00,0x18,0x18,0x18,0x18,0x18,0x00,0x18,0x18,0x18,0x18,0x18,0x00,0x00,0x00,0x00,0x00,0x70,0x18,0x18,0x18,0x0E,0x18,0x18,0x18,0x18,0x70,0x
00,0x00,0x00,0x00,0x00,0x00,0x3B,0x6E,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x70,0xD8,0xD8,0x70,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00}
```

… have fun!

edgarmarx@t-online.de