Graphics in pmode

'How do I enable graphics from protected mode?'

1. **You don't**. Graphics programming is fun, but graphics are hardly essential for an OS. **Don't get side-tracked**.
2. Call the BIOS mode-set interrupt **in the 16-bit boot code**, before the pmode kernel starts.
3. Use protected-mode code to **program the VGA directly**, without using the BIOS. This works only with VGA-compatible video boards and VGA-compatible video modes.
4. Add a **virtual 8086 mode monitor (VMM)** to your OS. Call the BIOS mode-set interrupt in virtual 8086 mode.
5. Switch from pmode to real mode, call the BIOS mode-set interrupt in real mode, then return to pmode.
6. **Write a protected-mode driver** specifically for the SVGA chip used in your video board. Someone else who wants to use your OS must have the same video board (or they must write a new driver for their own video board).
7. Call VBE 3.x BIOS functions in 16-bit protected mode. Few video cards support VBE 3.x.

Video timing

- Dot clock (pixel clock)
- Character clock = dot clock divided by 8 or 9
- Horizontal sync (retrace) frequency = character clock / horizontal total
- Horizontal total / horizontal displayed ~ 1.2 (20% overscan)
- Vertical sync (retrace) frequency = horizontal sync frequency / vertical total
- Vertical total / vertical displayed ~ 1.1 (10% overscan)

VGA

- INT 10h VGA BIOS interrupts which work in **real or virtual-8086 mode only**, to set mode, change font, etc.
- Dot clock is one of: 28.35, 25.2, 14.175, or 12.6 MHz
- Character clock is dot clock divided by 8 or 9
- Horizontal sync frequency for very old VGA monitors is **always 31.5 kHz**. Other frequencies can be produced by the VGA board and are supported by newer SVGA monitors.
- Vertical sync frequency for very old VGA monitors is **always 60 or 70 Hz**. Other frequencies can be produced by the VGA board and are supported by newer SVGA monitors.
- No interlace. No video memory 'banks'. No color depth > 8. Only 256K usable video memory max.
- Four memory *planes*, 64K each: P1, P2, P4, and P8
- **Text:** characters occupy even bytes in plane P1, attributes occupy odd bytes in plane P2, fonts are in plane P4. Text mode uses *Even/Odd* addressing: bit b0 of address selects between planes P1 and P2, so you need not change planes to set the attributes.
- **4-color (CGA) graphics**: 4 pixels per byte; uses planes P1 and P2. As with text, Even/Odd addressing eliminates the need to switch planes.
- **16-color graphics**: four monochrome planes: blue, green, red, and intense (the 16- and 256-color palettes can be re-defined to give you any 16 colors you want). Highest graphics resolution available with plain VGA, but very difficult to write software for this mode.
- **256-color graphics**: one byte per pixel. Simple software, but low resolution with plain VGA. All four planes are used with *Chain-4* addressing: bits b0 and b1 of the address select the plane so you need not worry about it. ***Mode-X*** is unchained 256-color mode: difficult to program, but it lets you use up to 256K video memory for 256-color modes up to 360x480 resolution.

Super VGA (SVGA)

- INT 10h AH=4Fh BIOS interrupts (VESA BIOS extensions; VBE). VBE 1.x runs in **real or virtual-8086 mode only**. VBE 2.x has a few not-too-useful pmode functions. For VBE 3.x, most functions (including mode-set) are carefully written to work in **real mode or 16-bit protected mode**.
- More than 256K video memory
- SVGA boards support many dot clock frequencies. SVGA monitors support many sync frequencies.
- 16-bit SVGA, VBE 1.x BIOS: video memory accessible one 64K 'bank' at a time. **Paging can be used to simulate a linear framebuffer**, but there are some 'gotchas' with this method (see links below)
- 32-bit SVGA, VBE 2.x or 3.x BIOS: ***Linear framebuffer*** for video memory -- no banks or planes, but video memory is no longer at A000:0000

Code snippets

VGA graphics demo. Pixels, rectangular fill, horizontal and vertical lines, and text blitting in 5 different graphics modes.
Set text or graphics video modes (including text font) **without using the BIOS**

TO DO

```
(S)VGA functional units:
- Miscellaneous register
- Sequencer (SEQ)
        - In unchained modes, SEQ register 2 selects one
          (or more!) planes to write
- CRT Controller (CRTC)
- Graphics Controller (GC)
        - In unchained modes, GC register 4 selects a plane to read
- Attribute Controller (AC)
        - 16-color palette in first 16 registers of AC
- 256-color palette (DAC)

Gotchas:
- 'Lock' bits in CRTC registers 3 and 17
- Screwy CGA addressing (disabled by bit b0 in CRTC register 23)
- Palette is not compact, i.e. for the 16-color palette
        entry[i] != i
```

VGA graphics demo - **grdemo.c**

```
/*****************************************************************************
VGA graphics demo
Chris Giese <geezer@execpc.com> http://my.execpc.com/~geezer
Release date: ?
This code is public domain (no copyright).
You can do whatever you want with it.

This code uses the BIOS to set graphics mode, and uses the BIOS font.
Should compile cleanly with Turbo C++ 1.0, Turbo C++ 3.0, 16- or 32-bit
Watcom C, or DJGPP. DJGPP version will not work in Windows NT/2000/XP
DOS box.

Some additional things you could do with this:
- Write a function tblit1(), similar to blit1(), that uses an on-off
  transparency mask. Use this function to blit non-rectangular objects
  such as a mouse cursor.
- Write blit_plane(): a fast function to blit from monochrome to monochrome
  or 4-plane bitmaps. Use an external shift() function, written in asm
- Support VBE 1.x banked framebuffer
- Support VBE 2.x linear framebuffer (pmode only, not at A000h:0000)
- Support greater color depths: 15 bpp, 16 bpp, 24 bpp, 32 bpp
- Color reduction, e.g. Heckbert (median-cut) algorithm
- Clipping engine that lets you draw a window that is partially
  obscured by "closer" windows
- Mouse, keyboard, and timer events
- Widgets: push button, checkbox, radio buttons, listbox, dialog, etc.
*****************************************************************************/
#include <string.h> /* [_f]memset() */
/******************************** TURBO C ************************************/
#if defined(__TURBOC__)
#include <dos.h> /* struct REGPACK, intr() */

/* The framebuffer is far outside the 16-bit data segment. The only way to
make the framebuffer work like in-memory bitmaps is to use far pointers.

We still use the SMALL memory model. */
#define       FAR             far
#define       FARPTR(S, O)    MK_FP(S, O)

#define       outportw(P,V)   outport(P,V)

#define       R_AX            r_ax
#define       R_BX            r_bx
#define       R_BP            r_bp
#define       R_ES            r_es

#define       trap(N,R)       intr(N,R)

typedef struct REGPACK regs_t;

#if __TURBOC__<0x300
void vmemset(unsigned char FAR *s, unsigned c, unsigned n)
{
      for(; n != 0; n--)
      {
              *s = c;
              s++;
      }
}
#else
void vmemset(unsigned char FAR *s, unsigned c, unsigned n)
{
      _fmemset(s, c, n);
}
#endif
```

```
/******************************** DJGPP ***********************************/
#elif defined(__DJGPP__)
#include <dpmi.h> /* __dpmi_... */
#include <dos.h> /* inportb(), outportb() */

#define       FAR             /* nothing */
#define       FARPTR(S, O)    (unsigned char *)((S) * 16L + (O) + \
                              __djgpp_conventional_base)

/* near pointers; not supported in Windows NT/2k/XP DOS box */
#include <sys/nearptr.h> /* __djgpp_conventional_base, __djgpp_nearptr_enable() */
#include <stdio.h> /* printf() */
#include <crt0.h> /* _CRT0_FLAG_NEARPTR, _crt0_startup_flags */

#define       R_AX            x.ax
#define       R_BX            x.bx
#define       R_BP            x.bp
#define       R_ES            x.es

#define       trap(N,R)       __dpmi_int(N,R)

typedef __dpmi_regs    regs_t;

void vmemset(unsigned char FAR *s, unsigned c, unsigned n)
{
      memset(s, c, n);
}

/****************************** WATCOM C ********************************/
#elif defined(__WATCOMC__)
#include <dos.h> /* union REGPACK, MK_FP(), intr() */

#if defined(__386__)
#define       FAR             /* nothing */
#define       FARPTR(S, O)    (unsigned char *)((S) * 16L + (O))

void vmemset(unsigned char FAR *s, unsigned c, unsigned n)
{
      memset(s, c, n);
}

#else
#define       FAR             far
#define       FARPTR(S, O)    MK_FP(S, O)

void vmemset(unsigned char FAR *s, unsigned c, unsigned n)
{
      _fmemset(s, c, n);
}
#endif

#define       inportb(P)      inp(P)
#define       outportb(P,V)   outp(P,V)
#define       outportw(P,V)   outpw(P,V)

#define       R_AX            w.ax
#define       R_BX            w.bx
#define       R_BP            w.bp
#define       R_ES            w.es

/* WARNING: for 32-bit code, unused fields of regs_t
must be zeroed before using this macro */
#define       trap(N,R)       intr(N,R)

typedef union REGPACK  regs_t;
```

```
#else
#error Not Turbo C, not DJGPP, not Watcom C. Sorry.
#endif

#include <conio.h> /* getch() */

/* need direct access to some VGA registers to select plane,
enable Mode X, and fix screwy CGA addressing */
#define VGA_SEQ_INDEX          0x3C4
#define VGA_SEQ_DATA           0x3C5
#define VGA_GC_INDEX           0x3CE
#define VGA_GC_DATA            0x3CF
#define VGA_CRTC_INDEX         0x3D4
#define VGA_CRTC_DATA          0x3D5

/* bitmap "class" */
typedef struct
{
      unsigned wd, ht;
      unsigned char FAR *raster;
      unsigned fore_color, back_color;
/* "member functions" */
      const struct _driver *ops;
} bmp_t;

typedef struct _driver
{
/* "pure virtual functions": color drivers MUST implement these */
      void (*write_pixel)(bmp_t *bmp, unsigned x, unsigned y, unsigned c);
      unsigned (*read_pixel)(bmp_t *bmp, unsigned x, unsigned y);
/* "virtual functions": drivers MAY implement these, for speed
fill rectangular area with solid color */
      void (*fill_rect)(bmp_t *bmp, int x, int y, int wd, int ht);
/* copy monochrome bitmap to this bitmap (used to display text) */
      void (*blit1)(bmp_t *src, bmp_t *dst, unsigned dst_x, unsigned dst_y);
/* copy all or part of one bitmap to another (both of the same depth) */
      void (*blit)(bmp_t *src, bmp_t *dst, unsigned dst_x, unsigned dst_y);
} ops_t;
/*============================================================================
helper functions
============================================================================*/
/*****************************************************************************
*****************************************************************************/
void set_plane(unsigned p)
{
      static unsigned curr_p = -1u;
/**/
      unsigned char pmask;

      p &= 3;
      if(p == curr_p)
               return;
      curr_p = p;
      pmask = 1 << p;
#if 0
      outportb(VGA_GC_INDEX, 4);
      outportb(VGA_GC_DATA, p);
      outportb(VGA_SEQ_INDEX, 2);
      outportb(VGA_SEQ_DATA, pmask);
#else
/* this is a little faster... */
      outportw(VGA_GC_INDEX, (p << 8) | 4);
      outportw(VGA_SEQ_INDEX, (pmask << 8) | 2);
#endif
}
```

```
/*************************************************************************
fast planar (monochrome or 16-color) rectangle fill
*************************************************************************/
void fill_plane(bmp_t *bmp, int x, int y, int wd, int ht, unsigned c)
{
      unsigned w, wd_in_bytes, off;
      unsigned char lmask, rmask;
      int x2, y2;

      x2 = x + wd - 1;
      w = (x2 >> 3) - (x >> 3) + 1;
      lmask = 0x00FF >> (x & 7); /* FF 7F 3F 1F 0F 07 03 01 */
      rmask = 0xFF80 >> (x2 & 7);/* 80 C0 E0 F0 F8 FC FE FF */
      if(w == 1)
               lmask &= rmask;
      wd_in_bytes = bmp->wd / 8;
      off = wd_in_bytes * y + x / 8;
      if(c)
/* for each row... */
               for(y2 = y; y2 < y + ht; y2++)
               {
/* do partial byte on left */
                        bmp->raster[off] |= lmask;
/* do solid bytes in middle */
                        if(w > 2)
                                 vmemset(bmp->raster + off + 1, 0xFF, w - 2);
/* do partial byte on right */
                        if(w > 1)
                                 bmp->raster[off + w - 1] |= rmask;
/* next row */
                        off += wd_in_bytes;
               }
      else
      {
               lmask = ~lmask;
               rmask = ~rmask;
               for(y2 = y; y2 < y + ht; y2++)
               {
                        bmp->raster[off] &= lmask;
                        if(w > 2)
                                 vmemset(bmp->raster + off + 1, 0, w - 2);
                        if(w > 1)
                                 bmp->raster[off + w - 1] &= rmask;
                        off += wd_in_bytes;
               }
      }
}
/*************************************************************************
fast planar blit
*************************************************************************/
void blit_plane(bmp_t *src, bmp_t *dst, unsigned dst_x, unsigned dst_y)
{
/* left as an exercise for the reader :)

You may need an external, assembly-language function to shift (left or
right) a long string of bytes. No need to shift by more than 7 bits. */
}
/*=======================================================================
driver for monochrome (1-bit) graphics
=======================================================================*/
/*************************************************************************
*************************************************************************/
static void write_pixel1(bmp_t *bmp, unsigned x, unsigned y, unsigned c)
{
      unsigned wd_in_bytes;
      unsigned off, mask;

      c = (c & 1) * 0xFF;
```

```
      wd_in_bytes = bmp->wd / 8;
      off = wd_in_bytes * y + x / 8;
      x = (x & 7) * 1;
      mask = 0x80 >> x;
      bmp->raster[off] = (bmp->raster[off] & ~mask) | (c & mask);
}
/*****************************************************************************
*****************************************************************************/
static unsigned read_pixel1(bmp_t *bmp, unsigned x, unsigned y)
{
      unsigned wd_in_bytes;
      unsigned off, mask;

      wd_in_bytes = bmp->wd / 8;
      off = wd_in_bytes * y + x / 8;
      x = (x & 7) * 1;
      mask = 0x80 >> x;
      return (bmp->raster[off] & mask) != 0;
}
/*****************************************************************************
*****************************************************************************/
static void fill_rect1(bmp_t *bmp, int x, int y, int wd, int ht)
{
      fill_plane(bmp, x, y, wd, ht, bmp->fore_color & 1);
}
/*****************************************************************************
*****************************************************************************/
const ops_t g_ops1 =
{
      write_pixel1,
      read_pixel1,
      fill_rect1,
      NULL, /* blit1 */
      NULL /* blit */
};
/*============================================================================
driver for 2-bit packed pixel (4-color CGA) graphics
============================================================================*/
/*****************************************************************************
*****************************************************************************/
static void write_pixel2(bmp_t *bmp, unsigned x, unsigned y, unsigned c)
{
      unsigned wd_in_bytes, off, mask;

      c = (c & 3) * 0x55;
      wd_in_bytes = bmp->wd / 4;
      off = wd_in_bytes * y + x / 4;
      x = (x & 3) * 2;
      mask = 0xC0 >> x;
      bmp->raster[off] = (bmp->raster[off] & ~mask) | (c & mask);
}
/*****************************************************************************
*****************************************************************************/
const ops_t g_ops2 =
{
      write_pixel2,
      NULL, /* read_pixel */
      NULL, /* fill_rect */
      NULL, /* blit1 */
      NULL /* blit */
};
```

```
/*============================================================================
driver for 4-plane 16-color graphics
============================================================================*/
/****************************************************************************
****************************************************************************/
static void write_pixel4p(bmp_t *bmp, unsigned x, unsigned y, unsigned c)
{
      unsigned wd_in_bytes, off, mask, p, pmask;

      wd_in_bytes = bmp->wd / 8;
      off = wd_in_bytes * y + x / 8;
      x = (x & 7) * 1;
      mask = 0x80 >> x;
      pmask = 1;
      for(p = 0; p < 4; p++)
      {
               set_plane(p);
               if(pmask & c)
                        bmp->raster[off] |= mask;
               else
                        bmp->raster[off] &= ~mask;
               pmask <<= 1;
      }
}
/****************************************************************************
pixel-by-pixel fill is too slow, so use this optimized function:
****************************************************************************/
static void fill_rect4p(bmp_t *bmp, int x, int y, int wd, int ht)
{
      unsigned char p, pmask;

      pmask = 1;
      for(p = 0; p < 4; p++)
      {
               set_plane(p);
               fill_plane(bmp, x, y, wd, ht, bmp->fore_color & pmask);
               pmask <<= 1;
      }
}
/****************************************************************************
****************************************************************************/
const ops_t g_ops4p =
{
      write_pixel4p,
      NULL, /* read_pixel */
      fill_rect4p,
      NULL, /* blit1 */
      NULL /* blit */
};
/*============================================================================
driver for 8-bit 256-color graphics
============================================================================*/
/****************************************************************************
****************************************************************************/
static void write_pixel8(bmp_t *bmp, unsigned x, unsigned y, unsigned c)
{
      unsigned wd_in_bytes;
      unsigned off;

      wd_in_bytes = bmp->wd;
      off = wd_in_bytes * y + x;
      bmp->raster[off] = c;
}
```

```
/*****************************************************************************
*****************************************************************************/
static void fill_rect8(bmp_t *bmp, int x, int y, int wd, int ht)
{
      unsigned wd_in_bytes, off, y2;

      wd_in_bytes = bmp->wd;
      off = wd_in_bytes * y + x;
      for(y2 = y; y2 < y + ht; y2++)
      {
               vmemset(bmp->raster + off, bmp->fore_color, wd);
               off += wd_in_bytes;
      }
}
/*****************************************************************************
*****************************************************************************/
const ops_t g_ops8 =
{
      write_pixel8,
      NULL, /* read_pixel */
      fill_rect8,
      NULL, /* blit1 */
      NULL /* blit */
};
/*===========================================================================
driver for 8-bit 256-color Mode-X graphics
===========================================================================*/
/*****************************************************************************
*****************************************************************************/
static void write_pixel8x(bmp_t *bmp, unsigned x, unsigned y, unsigned c)
{
      unsigned wd_in_bytes;
      unsigned off;

      wd_in_bytes = bmp->wd / 4;
      off = wd_in_bytes * y + x / 4;
      set_plane(x & 3);
      bmp->raster[off] = c;
}
/*****************************************************************************
*****************************************************************************/
const ops_t g_ops8x =
{
      write_pixel8x,
      NULL, /* read_pixel */
      NULL, /* fill_rect */
      NULL, /* blit1 */
      NULL /* blit */
};
/*===========================================================================
depth-independent routines, which call the depth-dependent routines
===========================================================================*/
/*****************************************************************************
*****************************************************************************/
unsigned read_pixel(bmp_t *bmp, unsigned x, unsigned y)
{
      if(x >= bmp->wd || y >= bmp->ht)
               return 0;
      if(bmp->ops->read_pixel == NULL)
               return 0; /* uh-oh */
      return bmp->ops->read_pixel(bmp, x, y);
}
```

```
/*************************************************************************
*************************************************************************/
void write_pixel(bmp_t *bmp, unsigned x, unsigned y, unsigned c)
{
	if(x >= bmp->wd || y >= bmp->ht)
		return;
	if(bmp->ops->write_pixel == NULL)
		return; /* uh-oh */
	bmp->ops->write_pixel(bmp, x, y, c);
}
/*************************************************************************
*************************************************************************/
void fill_rect(bmp_t *bmp, int x, int y, int wd, int ht)
{
	int x2, y2;

/* clip */
	if(x < 0)
	{
		if(wd + x < 0)
			return;
		wd += x;
		x = 0;
	}
	if(x + wd >= (int)bmp->wd)
	{
		if(x >= (int)bmp->wd)
			return;
		wd = bmp->wd - x;
	}
	if(y < 0)
	{
		if(ht + y < 0)
			return;
		ht += y;
		y = 0;
	}
	if(y + ht >= (int)bmp->ht)
	{
		if(y >= (int)bmp->ht)
			return;
		ht = bmp->ht - y;
	}
/* use fast routine if available */
	if(bmp->ops->fill_rect != NULL)
	{
		bmp->ops->fill_rect(bmp, x, y, wd, ht);
		return;
	}
	for(y2 = y; y2 < y + ht; y2++)
		for(x2 = x; x2 < x + wd; x2++)
			write_pixel(bmp, x2, y2, bmp->fore_color);
}
/*************************************************************************
*************************************************************************/
void hline(bmp_t *bmp, int x, int y, unsigned wd)
{
	fill_rect(bmp, x, y, wd, 1);
}
/*************************************************************************
*************************************************************************/
void vline(bmp_t *bmp, int x, int y, unsigned ht)
{
	fill_rect(bmp, x, y, 1, ht);
}
```

```
/*****************************************************************************
blit1 = blit from monochrome bitmap to bitmap of any color depth
*****************************************************************************/
void blit1(bmp_t *src, bmp_t *dst, unsigned dst_x, unsigned dst_y)
{
      unsigned x, y, c;

/* source bitmap _must_ be monochrome */
      if(src->ops != &g_ops1)
               return;
/* use fast routine if available */
      if(src->ops->blit1 != NULL)
      {
               src->ops->blit1(src, dst, dst_x, dst_y);
               return;
      }
      for(y = 0; y < src->ht; y++)
               for(x = 0; x < src->wd; x++)
               {
                        c = read_pixel(src, x, y);
/* xxx - on-off transparency?
                        if(c == 0)
                                 continue; */
                        if(c != 0)
                                 c = dst->fore_color;
                        else
                                 c = dst->back_color;
                        write_pixel(dst, dst_x + x, dst_y + y, c);
               }
}
/*****************************************************************************
blit = copy from one bitmap to another, both of the same color depth
*****************************************************************************/
void blit(bmp_t *src, bmp_t *dst, unsigned dst_x, unsigned dst_y)
{
      unsigned x, y, c;

/* they must be the same depth */
      if(src->ops != dst->ops)
               return;
/* use fast routine if available */
      if(src->ops->blit != NULL)
      {
               src->ops->blit(src, dst, dst_x, dst_y);
               return;
      }
      for(y = 0; y < src->ht; y++)
               for(x = 0; x < src->wd; x++)
               {
                        c = read_pixel(src, x, y);
                        write_pixel(dst, dst_x + x, dst_y + y, c);
               }
}
/*****************************************************************************
find 8x8 font in VGA BIOS ROM
*****************************************************************************/
unsigned char FAR *bios_8x8_font(void)
{
      unsigned char FAR *font;
      regs_t regs;

/* use BIOS INT 10h AX=1130h to find font #3 (8x8) in ROM */
      memset(&regs, 0, sizeof(regs)); /* for Watcom C */
      regs.R_AX = 0x1130;
      regs.R_BX = 0x0300;
      trap(0x10, &regs);
/* CauseWay DOS extender seems to return a selector in ES,
instead of real-mode segment value (usu. 0xC000) */
```

```
#if defined(__WATCOMC__)&&defined(__386__)
      font = FARPTR(0xC000, regs.R_BP);
#else
      font = FARPTR(regs.R_ES, regs.R_BP);
#endif
      return font;
}
/*****************************************************************************
*****************************************************************************/
void bputs(bmp_t *bmp, unsigned x, unsigned y, const char *s)
{
      unsigned char FAR *font;
      bmp_t src;

      font = bios_8x8_font();
      src.wd = 8;
      src.ht = 8;
      src.ops = &g_ops1;
      for(; *s != '\0'; s++)
      {
               src.raster = font + 8 * (*s);
               blit1(&src, bmp, x, y);
               x += 8;
      }
}
/*============================================================================
DEMO
============================================================================*/
/*****************************************************************************
*****************************************************************************/
static void border3d(bmp_t *bmp, int x, int y, unsigned wd, unsigned ht,
               char down)
{
      if(down)
      {
               bmp->fore_color = 8;
               hline(bmp, x + 0, y + 0, wd - 1);
               vline(bmp, x + 0, y + 0, ht - 1);
               bmp->fore_color = 0;
               hline(bmp, x + 1, y + 1, wd - 3);
               vline(bmp, x + 1, y + 1, ht - 3);
               bmp->fore_color = 7;
               hline(bmp, x + 1, y + ht - 2, wd - 2);
               vline(bmp, x + wd - 2, y + 1, ht - 2);
               bmp->fore_color = 15;
               hline(bmp, x + 0, y + ht - 1, wd);
               vline(bmp, x + wd - 1, y + 0, ht);
      }
      else
      {
               bmp->fore_color = 7;
               hline(bmp, x + 0, y + 0, wd - 1);
               vline(bmp, x + 0, y + 0, ht - 1);
               bmp->fore_color = 15;
               hline(bmp, x + 1, y + 1, wd - 3);
               vline(bmp, x + 1, y + 1, ht - 3);
               bmp->fore_color = 8;
               hline(bmp, x + 1, y + ht - 2, wd - 2);
               vline(bmp, x + wd - 2, y + 1, ht - 2);
               bmp->fore_color = 0;
               hline(bmp, x + 0, y + ht - 1, wd);
               vline(bmp, x + wd - 1, y + 0, ht);
      }
}
```

```
/****************************************************************************
****************************************************************************/
static void demo(bmp_t *bmp, const char *title)
{
      unsigned x = 10, y = 10, wd = 180, ht = 50;

/* erase screen to blue */
      bmp->fore_color = 1;
      fill_rect(bmp, 0, 0, bmp->wd, bmp->ht);
/* draw gray window with 3D border */
      bmp->fore_color = 7;
      fill_rect(bmp, x, y, wd, ht);
      border3d(bmp, x, y, wd, ht, 0);
/* draw white-on-green title bar */
      bmp->fore_color = 2;
      fill_rect(bmp, x + 2, y + 2, wd - 4, 10);
      bmp->back_color = 2;
      bmp->fore_color = 15;
      bputs(bmp, x + 3, y + 3, title);
/* draw menu bar on existing gray background */
      bmp->back_color = 7;
      bmp->fore_color = 0;
      bputs(bmp, x + 3, y + 13, "File Edit");
/* draw white inner area with 3D border */
      bmp->fore_color = 15;
      fill_rect(bmp, x + 3, y + 21, wd - 6, ht - 24);
      border3d(bmp, x + 3, y + 21, wd - 6, ht - 24, 1);
/* await key pressed */
      getch();
}
/****************************************************************************
****************************************************************************/
int main(void)
{
      static const unsigned wd[] =
      {
              640, 320, 640, 320, 320
      };
      static const unsigned ht[] =
      {
              480, 200, 480, 200, 200
      };
      static const ops_t *ops[] =
      {
              &g_ops1, &g_ops2, &g_ops4p, &g_ops8, &g_ops8x
      };
      static const unsigned mode[] =
      {
              0x11, 5, 0x12, 0x13, 0x13
      };
      static const char *title[] =
      {
        "640x480x2", "320x200x4", "640x480x16", "320x200x256", "320x200x256 ModeX"
      };
/**/
      regs_t regs;
      unsigned i;
      bmp_t bmp;

#if defined(__DJGPP__)
      if(!(_crt0_startup_flags & _CRT0_FLAG_NEARPTR))
      {
              if(!__djgpp_nearptr_enable())
              {
                      printf("Could not enable nearptr access "
                              "(Windows NT/2000/XP?)\n");
              }
      }
```

```
#endif
      for(i = 0; i < sizeof(wd) / sizeof(wd[0]); i++)
      {
               bmp.raster = FARPTR(0xA000, 0);
               bmp.wd = wd[i];
               bmp.ht = ht[i];
               bmp.ops = ops[i];

               memset(&regs, 0, sizeof(regs)); /* for Watcom C */
               regs.R_AX = mode[i];
               trap(0x10, &regs);
/* to make CGA graphics work like other graphics modes... */
               if(mode[i] == 0x05)
               {
/* 1) turn off screwy CGA addressing */
                        outportb(VGA_CRTC_INDEX, 0x17);
                        outportb(VGA_CRTC_DATA, inportb(VGA_CRTC_DATA) | 1);
/* 2) turn off doublescan */
                        outportb(VGA_CRTC_INDEX, 9);
                        outportb(VGA_CRTC_DATA, inportb(VGA_CRTC_DATA) & ~0x80);
/* 3) move the framebuffer from B800:0000 to A000:0000 */
                        outportb(VGA_GC_INDEX, 6);
                        outportb(VGA_GC_DATA, inportb(VGA_GC_INDEX) & ~0x0C);
               }
/* to convert mode 13h to Mode X... */
               else if(i == 4)
               {
/* 1) turn off Chain-4 addressing */
                        outportb(VGA_SEQ_INDEX, 0x04);
                        outportb(VGA_SEQ_DATA, inportb(VGA_SEQ_DATA) & ~0x08);
/* 2) turn off doubleword clocking */
                        outportb(VGA_CRTC_INDEX, 0x14);
                        outportb(VGA_CRTC_DATA, inportb(VGA_CRTC_DATA) & ~0x40);
/* 3) turn off word clocking in case it's on */
                        outportb(VGA_CRTC_INDEX, 0x17);
                        outportb(VGA_CRTC_DATA, inportb(VGA_CRTC_DATA) | 0x40);
               }
               demo(&bmp, title[i]);
      }
/* return to text mode */
      memset(&regs, 0, sizeof(regs)); /* for Watcom C */
      regs.R_AX = 0x03;
      trap(0x10, &regs);
      return 0;
}
```

Set text or graphics video modes (including text font) **without using the BIOS - modes.c**

```
/*****************************************************************************
Sets VGA-compatible video modes without using the BIOS
Chris Giese <geezer@execpc.com>   http://my.execpc.com/~geezer
Release date: ?
This code is public domain (no copyright).
You can do whatever you want with it.

To do:
- more registers dumps, for various text modes and ModeX
- flesh out code to support SVGA chips?
- do something with 16- and 256-color palettes?
*****************************************************************************/
#include <string.h> /* movedata(), memcpy() */
#include <stdio.h> /* printf() */
#include <conio.h> /* getch() */
#include <dos.h> /* FP_SEG(), FP_OFF(), inportb(), outportb() */

/******************************** TURBO C ********************************/
#if defined(__TURBOC__)

#define  pokew(S,O,V)              poke(S,O,V)
#define  _vmemwr(DS,DO,S,N)        movedata(FP_SEG(S), FP_OFF(S), DS, DO, N)

/********************************* DJGPP *********************************/
#elif defined(__DJGPP__)
#include <sys/movedata.h> /* dosmemput() */
#include <sys/farptr.h> /* _farpoke[b|w]() */
#include <go32.h> /* _dos_ds */

#define  peekb(S,O)                _farpeekb(_dos_ds, 16uL * (S) + (O))
#define  pokeb(S,O,V)              _farpokeb(_dos_ds, 16uL * (S) + (O), V)
#define  pokew(S,O,V)              _farpokew(_dos_ds, 16uL * (S) + (O), V)
#define  _vmemwr(DS,DO,S,N)        dosmemput(S, N, 16uL * (DS) + (DO))

/******************************** WATCOM C *******************************/
#elif defined(__WATCOMC__)
#include <conio.h> /* inp(), outp() */

#define  inportb(P)                inp(P)
#define  outportb(P,V)             outp(P,V)

#if defined(__386__)
/* CauseWay DOS extender only */
#define  peekb(S,O)                *(unsigned char *)(16uL * (S) + (O))
#define  pokeb(S,O,V)              *(unsigned char *)(16uL * (S) + (O)) = (V)
#define  pokew(S,O,V)              *(unsigned short *)(16uL * (S) + (O)) = (V)
#define  _vmemwr(DS,DO,S,N)        memcpy((char *)((DS) * 16 + (DO)), S, N)

#else
#define  peekb(S,O)                *(unsigned char far *)MK_FP(S, O)
#define  pokeb(S,O,V)              *(unsigned char far *)MK_FP(S, O) = (V)
#define  pokew(S,O,V)              *(unsigned short far *)MK_FP(S, O) = (V)
#define  _vmemwr(DS,DO,S,N)        movedata(FP_SEG(S), FP_OFF(S), DS, DO, N)
#endif

/*****************************************************************************/
#else
#error Unsupported compiler.
#endif

#define  VGA_AC_INDEX              0x3C0
#define  VGA_AC_WRITE              0x3C0
#define  VGA_AC_READ               0x3C1
#define  VGA_MISC_WRITE            0x3C2
#define VGA_SEQ_INDEX              0x3C4
#define VGA_SEQ_DATA               0x3C5
```

```
#define  VGA_DAC_READ_INDEX       0x3C7
#define  VGA_DAC_WRITE_INDEX      0x3C8
#define  VGA_DAC_DATA             0x3C9
#define  VGA_MISC_READ            0x3CC
#define VGA_GC_INDEX              0x3CE
#define VGA_GC_DATA               0x3CF
/*                          COLOR emulation          MONO emulation */
#define VGA_CRTC_INDEX            0x3D4            /* 0x3B4 */
#define VGA_CRTC_DATA             0x3D5            /* 0x3B5 */
#define  VGA_INSTAT_READ          0x3DA

#define  VGA_NUM_SEQ_REGS 5
#define  VGA_NUM_CRTC_REGS        25
#define  VGA_NUM_GC_REGS          9
#define  VGA_NUM_AC_REGS          21
#define  VGA_NUM_REGS             (1 + VGA_NUM_SEQ_REGS + VGA_NUM_CRTC_REGS + \
                                  VGA_NUM_GC_REGS + VGA_NUM_AC_REGS)
/*****************************************************************************
VGA REGISTER DUMPS FOR VARIOUS TEXT MODES

()=to do
        40x25    (40x30)  40x50    (40x60)
        (45x25)  (45x30)  (45x50)  (45x60)
        80x25    (80x30)  80x50    (80x60)
        (90x25)  90x30    (90x50)  90x60
*****************************************************************************/
unsigned char g_40x25_text[] =
{
/* MISC */
        0x67,
/* SEQ */
        0x03, 0x08, 0x03, 0x00, 0x02,
/* CRTC */
        0x2D, 0x27, 0x28, 0x90, 0x2B, 0xA0, 0xBF, 0x1F,
        0x00, 0x4F, 0x0D, 0x0E, 0x00, 0x00, 0x00, 0xA0,
        0x9C, 0x8E, 0x8F, 0x14, 0x1F, 0x96, 0xB9, 0xA3,
        0xFF,
/* GC */
        0x00, 0x00, 0x00, 0x00, 0x00, 0x10, 0x0E, 0x00,
        0xFF,
/* AC */
        0x00, 0x01, 0x02, 0x03, 0x04, 0x05, 0x14, 0x07,
        0x38, 0x39, 0x3A, 0x3B, 0x3C, 0x3D, 0x3E, 0x3F,
        0x0C, 0x00, 0x0F, 0x08, 0x00,
};

unsigned char g_40x50_text[] =
{
/* MISC */
        0x67,
/* SEQ */
        0x03, 0x08, 0x03, 0x00, 0x02,
/* CRTC */
        0x2D, 0x27, 0x28, 0x90, 0x2B, 0xA0, 0xBF, 0x1F,
        0x00, 0x47, 0x06, 0x07, 0x00, 0x00, 0x04, 0x60,
        0x9C, 0x8E, 0x8F, 0x14, 0x1F, 0x96, 0xB9, 0xA3,
        0xFF,
/* GC */
        0x00, 0x00, 0x00, 0x00, 0x00, 0x10, 0x0E, 0x00,
        0xFF,
/* AC */
        0x00, 0x01, 0x02, 0x03, 0x04, 0x05, 0x14, 0x07,
        0x38, 0x39, 0x3A, 0x3B, 0x3C, 0x3D, 0x3E, 0x3F,
        0x0C, 0x00, 0x0F, 0x08, 0x00,
};

unsigned char g_80x25_text[] =
{
```

```
/* MISC */
        0x67,
/* SEQ */
        0x03, 0x00, 0x03, 0x00, 0x02,
/* CRTC */
        0x5F, 0x4F, 0x50, 0x82, 0x55, 0x81, 0xBF, 0x1F,
        0x00, 0x4F, 0x0D, 0x0E, 0x00, 0x00, 0x00, 0x50,
        0x9C, 0x0E, 0x8F, 0x28, 0x1F, 0x96, 0xB9, 0xA3,
        0xFF,
/* GC */
        0x00, 0x00, 0x00, 0x00, 0x00, 0x10, 0x0E, 0x00,
        0xFF,
/* AC */
        0x00, 0x01, 0x02, 0x03, 0x04, 0x05, 0x14, 0x07,
        0x38, 0x39, 0x3A, 0x3B, 0x3C, 0x3D, 0x3E, 0x3F,
        0x0C, 0x00, 0x0F, 0x08, 0x00
};

unsigned char g_80x50_text[] =
{
/* MISC */
        0x67,
/* SEQ */
        0x03, 0x00, 0x03, 0x00, 0x02,
/* CRTC */
        0x5F, 0x4F, 0x50, 0x82, 0x55, 0x81, 0xBF, 0x1F,
        0x00, 0x47, 0x06, 0x07, 0x00, 0x00, 0x01, 0x40,
        0x9C, 0x8E, 0x8F, 0x28, 0x1F, 0x96, 0xB9, 0xA3,
        0xFF,
/* GC */
        0x00, 0x00, 0x00, 0x00, 0x00, 0x10, 0x0E, 0x00,
        0xFF,
/* AC */
        0x00, 0x01, 0x02, 0x03, 0x04, 0x05, 0x14, 0x07,
        0x38, 0x39, 0x3A, 0x3B, 0x3C, 0x3D, 0x3E, 0x3F,
        0x0C, 0x00, 0x0F, 0x08, 0x00,
};

unsigned char g_90x30_text[] =
{
/* MISC */
        0xE7,
/* SEQ */
        0x03, 0x01, 0x03, 0x00, 0x02,
/* CRTC */
        0x6B, 0x59, 0x5A, 0x82, 0x60, 0x8D, 0x0B, 0x3E,
        0x00, 0x4F, 0x0D, 0x0E, 0x00, 0x00, 0x00, 0x00,
        0xEA, 0x0C, 0xDF, 0x2D, 0x10, 0xE8, 0x05, 0xA3,
        0xFF,
/* GC */
        0x00, 0x00, 0x00, 0x00, 0x00, 0x10, 0x0E, 0x00,
        0xFF,
/* AC */
        0x00, 0x01, 0x02, 0x03, 0x04, 0x05, 0x14, 0x07,
        0x38, 0x39, 0x3A, 0x3B, 0x3C, 0x3D, 0x3E, 0x3F,
        0x0C, 0x00, 0x0F, 0x08, 0x00,
};

unsigned char g_90x60_text[] =
{
/* MISC */
        0xE7,
/* SEQ */
        0x03, 0x01, 0x03, 0x00, 0x02,
/* CRTC */
        0x6B, 0x59, 0x5A, 0x82, 0x60, 0x8D, 0x0B, 0x3E,
        0x00, 0x47, 0x06, 0x07, 0x00, 0x00, 0x00, 0x00,
        0xEA, 0x0C, 0xDF, 0x2D, 0x08, 0xE8, 0x05, 0xA3,
```

```
        0xFF,
/* GC */
        0x00, 0x00, 0x00, 0x00, 0x00, 0x10, 0x0E, 0x00,
        0xFF,
/* AC */
        0x00, 0x01, 0x02, 0x03, 0x04, 0x05, 0x14, 0x07,
        0x38, 0x39, 0x3A, 0x3B, 0x3C, 0x3D, 0x3E, 0x3F,
        0x0C, 0x00, 0x0F, 0x08, 0x00,
};
/****************************************************************************
VGA REGISTER DUMPS FOR VARIOUS GRAPHICS MODES
****************************************************************************/
unsigned char g_640x480x2[] =
{
/* MISC */
        0xE3,
/* SEQ */
        0x03, 0x01, 0x0F, 0x00, 0x06,
/* CRTC */
        0x5F, 0x4F, 0x50, 0x82, 0x54, 0x80, 0x0B, 0x3E,
        0x00, 0x40, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
        0xEA, 0x0C, 0xDF, 0x28, 0x00, 0xE7, 0x04, 0xE3,
        0xFF,
/* GC */
        0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x05, 0x0F,
        0xFF,
/* AC */
        0x00, 0x01, 0x02, 0x03, 0x04, 0x05, 0x14, 0x07,
        0x38, 0x39, 0x3A, 0x3B, 0x3C, 0x3D, 0x3E, 0x3F,
        0x01, 0x00, 0x0F, 0x00, 0x00
};
/****************************************************************************
*** NOTE: the mode described by g_320x200x4[]
is different from BIOS mode 05h in two ways:
- Framebuffer is at A000:0000 instead of B800:0000
- Framebuffer is linear (no screwy line-by-line CGA addressing)
****************************************************************************/
unsigned char g_320x200x4[] =
{
/* MISC */
        0x63,
/* SEQ */
        0x03, 0x09, 0x03, 0x00, 0x02,
/* CRTC */
        0x2D, 0x27, 0x28, 0x90, 0x2B, 0x80, 0xBF, 0x1F,
        0x00, 0x41, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
        0x9C, 0x0E, 0x8F, 0x14, 0x00, 0x96, 0xB9, 0xA3,
        0xFF,
/* GC */
        0x00, 0x00, 0x00, 0x00, 0x00, 0x30, 0x02, 0x00,
        0xFF,
/* AC */
        0x00, 0x13, 0x15, 0x17, 0x02, 0x04, 0x06, 0x07,
        0x10, 0x11, 0x12, 0x13, 0x14, 0x15, 0x16, 0x17,
        0x01, 0x00, 0x03, 0x00, 0x00
};

unsigned char g_640x480x16[] =
{
/* MISC */
        0xE3,
/* SEQ */
        0x03, 0x01, 0x08, 0x00, 0x06,
/* CRTC */
        0x5F, 0x4F, 0x50, 0x82, 0x54, 0x80, 0x0B, 0x3E,
        0x00, 0x40, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
        0xEA, 0x0C, 0xDF, 0x28, 0x00, 0xE7, 0x04, 0xE3,
        0xFF,
```

```
/* GC */
        0x00, 0x00, 0x00, 0x00, 0x03, 0x00, 0x05, 0x0F,
        0xFF,
/* AC */
        0x00, 0x01, 0x02, 0x03, 0x04, 0x05, 0x14, 0x07,
        0x38, 0x39, 0x3A, 0x3B, 0x3C, 0x3D, 0x3E, 0x3F,
        0x01, 0x00, 0x0F, 0x00, 0x00
};

unsigned char g_720x480x16[] =
{
/* MISC */
        0xE7,
/* SEQ */
        0x03, 0x01, 0x08, 0x00, 0x06,
/* CRTC */
        0x6B, 0x59, 0x5A, 0x82, 0x60, 0x8D, 0x0B, 0x3E,
        0x00, 0x40, 0x06, 0x07, 0x00, 0x00, 0x00, 0x00,
        0xEA, 0x0C, 0xDF, 0x2D, 0x08, 0xE8, 0x05, 0xE3,
        0xFF,
/* GC */
        0x00, 0x00, 0x00, 0x00, 0x03, 0x00, 0x05, 0x0F,
        0xFF,
/* AC */
        0x00, 0x01, 0x02, 0x03, 0x04, 0x05, 0x06, 0x07,
        0x08, 0x09, 0x0A, 0x0B, 0x0C, 0x0D, 0x0E, 0x0F,
        0x01, 0x00, 0x0F, 0x00, 0x00,
};

unsigned char g_320x200x256[] =
{
/* MISC */
        0x63,
/* SEQ */
        0x03, 0x01, 0x0F, 0x00, 0x0E,
/* CRTC */
        0x5F, 0x4F, 0x50, 0x82, 0x54, 0x80, 0xBF, 0x1F,
        0x00, 0x41, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
        0x9C, 0x0E, 0x8F, 0x28,   0x40, 0x96, 0xB9, 0xA3,
        0xFF,
/* GC */
        0x00, 0x00, 0x00, 0x00, 0x00, 0x40, 0x05, 0x0F,
        0xFF,
/* AC */
        0x00, 0x01, 0x02, 0x03, 0x04, 0x05, 0x06, 0x07,
        0x08, 0x09, 0x0A, 0x0B, 0x0C, 0x0D, 0x0E, 0x0F,
        0x41, 0x00, 0x0F, 0x00,   0x00
};

unsigned char g_320x200x256_modex[] =
{
/* MISC */
        0x63,
/* SEQ */
        0x03, 0x01, 0x0F, 0x00, 0x06,
/* CRTC */
        0x5F, 0x4F, 0x50, 0x82, 0x54, 0x80, 0xBF, 0x1F,
        0x00, 0x41, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
        0x9C, 0x0E, 0x8F, 0x28, 0x00, 0x96, 0xB9, 0xE3,
        0xFF,
/* GC */
        0x00, 0x00, 0x00, 0x00, 0x00, 0x40, 0x05, 0x0F,
        0xFF,
/* AC */
        0x00, 0x01, 0x02, 0x03, 0x04, 0x05, 0x06, 0x07,
        0x08, 0x09, 0x0A, 0x0B, 0x0C, 0x0D, 0x0E, 0x0F,
        0x41, 0x00, 0x0F, 0x00, 0x00
};
```

```
/* g_360x480x256_modex - to do */

/*****************************************************************************
8X8 AND 8X16 FONTS
*****************************************************************************/
static unsigned char g_8x8_font[2048] =
{
        0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
        0x7E, 0x81, 0xA5, 0x81, 0xBD, 0x99, 0x81, 0x7E,
        0x7E, 0xFF, 0xDB, 0xFF, 0xC3, 0xE7, 0xFF, 0x7E,
        0x6C, 0xFE, 0xFE, 0xFE, 0x7C, 0x38, 0x10, 0x00,
        0x10, 0x38, 0x7C, 0xFE, 0x7C, 0x38, 0x10, 0x00,
        0x38, 0x7C, 0x38, 0xFE, 0xFE, 0x92, 0x10, 0x7C,
        0x00, 0x10, 0x38, 0x7C, 0xFE, 0x7C, 0x38, 0x7C,
        0x00, 0x00, 0x18, 0x3C, 0x3C, 0x18, 0x00, 0x00,
        0xFF, 0xFF, 0xE7, 0xC3, 0xC3, 0xE7, 0xFF, 0xFF,
        0x00, 0x3C, 0x66, 0x42, 0x42, 0x66, 0x3C, 0x00,
        0xFF, 0xC3, 0x99, 0xBD, 0xBD, 0x99, 0xC3, 0xFF,
        0x0F, 0x07, 0x0F, 0x7D, 0xCC, 0xCC, 0xCC, 0x78,
        0x3C, 0x66, 0x66, 0x66, 0x3C, 0x18, 0x7E, 0x18,
        0x3F, 0x33, 0x3F, 0x30, 0x30, 0x70, 0xF0, 0xE0,
        0x7F, 0x63, 0x7F, 0x63, 0x63, 0x67, 0xE6, 0xC0,
        0x99, 0x5A, 0x3C, 0xE7, 0xE7, 0x3C, 0x5A, 0x99,
        0x80, 0xE0, 0xF8, 0xFE, 0xF8, 0xE0, 0x80, 0x00,
        0x02, 0x0E, 0x3E, 0xFE, 0x3E, 0x0E, 0x02, 0x00,
        0x18, 0x3C, 0x7E, 0x18, 0x18, 0x7E, 0x3C, 0x18,
        0x66, 0x66, 0x66, 0x66, 0x66, 0x00, 0x66, 0x00,
        0x7F, 0xDB, 0xDB, 0x7B, 0x1B, 0x1B, 0x1B, 0x00,
        0x3E, 0x63, 0x38, 0x6C, 0x6C, 0x38, 0x86, 0xFC,
        0x00, 0x00, 0x00, 0x00, 0x7E, 0x7E, 0x7E, 0x00,
        0x18, 0x3C, 0x7E, 0x18, 0x7E, 0x3C, 0x18, 0xFF,
        0x18, 0x3C, 0x7E, 0x18, 0x18, 0x18, 0x18, 0x00,
        0x18, 0x18, 0x18, 0x18, 0x7E, 0x3C, 0x18, 0x00,
        0x00, 0x18, 0x0C, 0xFE, 0x0C, 0x18, 0x00, 0x00,
        0x00, 0x30, 0x60, 0xFE, 0x60, 0x30, 0x00, 0x00,
        0x00, 0x00, 0xC0, 0xC0, 0xC0, 0xFE, 0x00, 0x00,
        0x00, 0x24, 0x66, 0xFF, 0x66, 0x24, 0x00, 0x00,
        0x00, 0x18, 0x3C, 0x7E, 0xFF, 0xFF, 0x00, 0x00,
        0x00, 0xFF, 0xFF, 0x7E, 0x3C, 0x18, 0x00, 0x00,
        0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
        0x18, 0x3C, 0x3C, 0x18, 0x18, 0x00, 0x18, 0x00,
        0x6C, 0x6C, 0x6C, 0x00, 0x00, 0x00, 0x00, 0x00,
        0x6C, 0x6C, 0xFE, 0x6C, 0xFE, 0x6C, 0x6C, 0x00,
        0x18, 0x7E, 0xC0, 0x7C, 0x06, 0xFC, 0x18, 0x00,
        0x00, 0xC6, 0xCC, 0x18, 0x30, 0x66, 0xC6, 0x00,
        0x38, 0x6C, 0x38, 0x76, 0xDC, 0xCC, 0x76, 0x00,
        0x30, 0x30, 0x60, 0x00, 0x00, 0x00, 0x00, 0x00,
        0x18, 0x30, 0x60, 0x60, 0x60, 0x30, 0x18, 0x00,
        0x60, 0x30, 0x18, 0x18, 0x18, 0x30, 0x60, 0x00,
        0x00, 0x66, 0x3C, 0xFF, 0x3C, 0x66, 0x00, 0x00,
        0x00, 0x18, 0x18, 0x7E, 0x18, 0x18, 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,
        0x06, 0x0C, 0x18, 0x30, 0x60, 0xC0, 0x80, 0x00,
        0x7C, 0xCE, 0xDE, 0xF6, 0xE6, 0xC6, 0x7C, 0x00,
        0x30, 0x70, 0x30, 0x30, 0x30, 0x30, 0xFC, 0x00,
        0x78, 0xCC, 0x0C, 0x38, 0x60, 0xCC, 0xFC, 0x00,
        0x78, 0xCC, 0x0C, 0x38, 0x0C, 0xCC, 0x78, 0x00,
        0x1C, 0x3C, 0x6C, 0xCC, 0xFE, 0x0C, 0x1E, 0x00,
        0xFC, 0xC0, 0xF8, 0x0C, 0x0C, 0xCC, 0x78, 0x00,
        0x38, 0x60, 0xC0, 0xF8, 0xCC, 0xCC, 0x78, 0x00,
        0xFC, 0xCC, 0x0C, 0x18, 0x30, 0x30, 0x30, 0x00,
        0x78, 0xCC, 0xCC, 0x78, 0xCC, 0xCC, 0x78, 0x00,
        0x78, 0xCC, 0xCC, 0x7C, 0x0C, 0x18, 0x70, 0x00,
        0x00, 0x18, 0x18, 0x00, 0x00, 0x18, 0x18, 0x00,
        0x00, 0x18, 0x18, 0x00, 0x00, 0x18, 0x18, 0x30,
```

```
0x18, 0x30, 0x60, 0xC0, 0x60, 0x30, 0x18, 0x00,
0x00, 0x00, 0x7E, 0x00, 0x7E, 0x00, 0x00, 0x00,
0x60, 0x30, 0x18, 0x0C, 0x18, 0x30, 0x60, 0x00,
0x3C, 0x66, 0x0C, 0x18, 0x18, 0x00, 0x18, 0x00,
0x7C, 0xC6, 0xDE, 0xDE, 0xDC, 0xC0, 0x7C, 0x00,
0x30, 0x78, 0xCC, 0xCC, 0xFC, 0xCC, 0xCC, 0x00,
0xFC, 0x66, 0x66, 0x7C, 0x66, 0x66, 0xFC, 0x00,
0x3C, 0x66, 0xC0, 0xC0, 0xC0, 0x66, 0x3C, 0x00,
0xF8, 0x6C, 0x66, 0x66, 0x66, 0x6C, 0xF8, 0x00,
0xFE, 0x62, 0x68, 0x78, 0x68, 0x62, 0xFE, 0x00,
0xFE, 0x62, 0x68, 0x78, 0x68, 0x60, 0xF0, 0x00,
0x3C, 0x66, 0xC0, 0xC0, 0xCE, 0x66, 0x3A, 0x00,
0xCC, 0xCC, 0xCC, 0xFC, 0xCC, 0xCC, 0xCC, 0x00,
0x78, 0x30, 0x30, 0x30, 0x30, 0x30, 0x78, 0x00,
0x1E, 0x0C, 0x0C, 0x0C, 0xCC, 0xCC, 0x78, 0x00,
0xE6, 0x66, 0x6C, 0x78, 0x6C, 0x66, 0xE6, 0x00,
0xF0, 0x60, 0x60, 0x60, 0x62, 0x66, 0xFE, 0x00,
0xC6, 0xEE, 0xFE, 0xFE, 0xD6, 0xC6, 0xC6, 0x00,
0xC6, 0xE6, 0xF6, 0xDE, 0xCE, 0xC6, 0xC6, 0x00,
0x38, 0x6C, 0xC6, 0xC6, 0xC6, 0x6C, 0x38, 0x00,
0xFC, 0x66, 0x66, 0x7C, 0x60, 0x60, 0xF0, 0x00,
0x7C, 0xC6, 0xC6, 0xC6, 0xD6, 0x7C, 0x0E, 0x00,
0xFC, 0x66, 0x66, 0x7C, 0x6C, 0x66, 0xE6, 0x00,
0x7C, 0xC6, 0xE0, 0x78, 0x0E, 0xC6, 0x7C, 0x00,
0xFC, 0xB4, 0x30, 0x30, 0x30, 0x30, 0x78, 0x00,
0xCC, 0xCC, 0xCC, 0xCC, 0xCC, 0xCC, 0xFC, 0x00,
0xCC, 0xCC, 0xCC, 0xCC, 0xCC, 0x78, 0x30, 0x00,
0xC6, 0xC6, 0xC6, 0xC6, 0xD6, 0xFE, 0x6C, 0x00,
0xC6, 0xC6, 0x6C, 0x38, 0x6C, 0xC6, 0xC6, 0x00,
0xCC, 0xCC, 0xCC, 0x78, 0x30, 0x30, 0x78, 0x00,
0xFE, 0xC6, 0x8C, 0x18, 0x32, 0x66, 0xFE, 0x00,
0x78, 0x60, 0x60, 0x60, 0x60, 0x60, 0x78, 0x00,
0xC0, 0x60, 0x30, 0x18, 0x0C, 0x06, 0x02, 0x00,
0x78, 0x18, 0x18, 0x18, 0x18, 0x18, 0x78, 0x00,
0x10, 0x38, 0x6C, 0xC6, 0x00, 0x00, 0x00, 0x00,
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0xFF,
0x30, 0x30, 0x18, 0x00, 0x00, 0x00, 0x00, 0x00,
0x00, 0x00, 0x78, 0x0C, 0x7C, 0xCC, 0x76, 0x00,
0xE0, 0x60, 0x60, 0x7C, 0x66, 0x66, 0xDC, 0x00,
0x00, 0x00, 0x78, 0xCC, 0xC0, 0xCC, 0x78, 0x00,
0x1C, 0x0C, 0x0C, 0x7C, 0xCC, 0xCC, 0x76, 0x00,
0x00, 0x00, 0x78, 0xCC, 0xFC, 0xC0, 0x78, 0x00,
0x38, 0x6C, 0x64, 0xF0, 0x60, 0x60, 0xF0, 0x00,
0x00, 0x00, 0x76, 0xCC, 0xCC, 0x7C, 0x0C, 0xF8,
0xE0, 0x60, 0x6C, 0x76, 0x66, 0x66, 0xE6, 0x00,
0x30, 0x00, 0x70, 0x30, 0x30, 0x30, 0x78, 0x00,
0x0C, 0x00, 0x1C, 0x0C, 0x0C, 0xCC, 0xCC, 0x78,
0xE0, 0x60, 0x66, 0x6C, 0x78, 0x6C, 0xE6, 0x00,
0x70, 0x30, 0x30, 0x30, 0x30, 0x30, 0x78, 0x00,
0x00, 0x00, 0xCC, 0xFE, 0xFE, 0xD6, 0xD6, 0x00,
0x00, 0x00, 0xB8, 0xCC, 0xCC, 0xCC, 0xCC, 0x00,
0x00, 0x00, 0x78, 0xCC, 0xCC, 0xCC, 0x78, 0x00,
0x00, 0x00, 0xDC, 0x66, 0x66, 0x7C, 0x60, 0xF0,
0x00, 0x00, 0x76, 0xCC, 0xCC, 0x7C, 0x0C, 0x1E,
0x00, 0x00, 0xDC, 0x76, 0x62, 0x60, 0xF0, 0x00,
0x00, 0x00, 0x7C, 0xC0, 0x70, 0x1C, 0xF8, 0x00,
0x10, 0x30, 0xFC, 0x30, 0x30, 0x34, 0x18, 0x00,
0x00, 0x00, 0xCC, 0xCC, 0xCC, 0xCC, 0x76, 0x00,
0x00, 0x00, 0xCC, 0xCC, 0xCC, 0x78, 0x30, 0x00,
0x00, 0x00, 0xC6, 0xC6, 0xD6, 0xFE, 0x6C, 0x00,
0x00, 0x00, 0xC6, 0x6C, 0x38, 0x6C, 0xC6, 0x00,
0x00, 0x00, 0xCC, 0xCC, 0xCC, 0x7C, 0x0C, 0xF8,
0x00, 0x00, 0xFC, 0x98, 0x30, 0x64, 0xFC, 0x00,
0x1C, 0x30, 0x30, 0xE0, 0x30, 0x30, 0x1C, 0x00,
0x18, 0x18, 0x18, 0x00, 0x18, 0x18, 0x18, 0x00,
0xE0, 0x30, 0x30, 0x1C, 0x30, 0x30, 0xE0, 0x00,
0x76, 0xDC, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
0x00, 0x10, 0x38, 0x6C, 0xC6, 0xC6, 0xFE, 0x00,
```

```
0x7C, 0xC6, 0xC0, 0xC6, 0x7C, 0x0C, 0x06, 0x7C,
0x00, 0xCC, 0x00, 0xCC, 0xCC, 0xCC, 0x76, 0x00,
0x1C, 0x00, 0x78, 0xCC, 0xFC, 0xC0, 0x78, 0x00,
0x7E, 0x81, 0x3C, 0x06, 0x3E, 0x66, 0x3B, 0x00,
0xCC, 0x00, 0x78, 0x0C, 0x7C, 0xCC, 0x76, 0x00,
0xE0, 0x00, 0x78, 0x0C, 0x7C, 0xCC, 0x76, 0x00,
0x30, 0x30, 0x78, 0x0C, 0x7C, 0xCC, 0x76, 0x00,
0x00, 0x00, 0x7C, 0xC6, 0xC0, 0x78, 0x0C, 0x38,
0x7E, 0x81, 0x3C, 0x66, 0x7E, 0x60, 0x3C, 0x00,
0xCC, 0x00, 0x78, 0xCC, 0xFC, 0xC0, 0x78, 0x00,
0xE0, 0x00, 0x78, 0xCC, 0xFC, 0xC0, 0x78, 0x00,
0xCC, 0x00, 0x70, 0x30, 0x30, 0x30, 0x78, 0x00,
0x7C, 0x82, 0x38, 0x18, 0x18, 0x18, 0x3C, 0x00,
0xE0, 0x00, 0x70, 0x30, 0x30, 0x30, 0x78, 0x00,
0xC6, 0x10, 0x7C, 0xC6, 0xFE, 0xC6, 0xC6, 0x00,
0x30, 0x30, 0x00, 0x78, 0xCC, 0xFC, 0xCC, 0x00,
0x1C, 0x00, 0xFC, 0x60, 0x78, 0x60, 0xFC, 0x00,
0x00, 0x00, 0x7F, 0x0C, 0x7F, 0xCC, 0x7F, 0x00,
0x3E, 0x6C, 0xCC, 0xFE, 0xCC, 0xCC, 0xCE, 0x00,
0x78, 0x84, 0x00, 0x78, 0xCC, 0xCC, 0x78, 0x00,
0x00, 0xCC, 0x00, 0x78, 0xCC, 0xCC, 0x78, 0x00,
0x00, 0xE0, 0x00, 0x78, 0xCC, 0xCC, 0x78, 0x00,
0x78, 0x84, 0x00, 0xCC, 0xCC, 0xCC, 0x76, 0x00,
0x00, 0xE0, 0x00, 0xCC, 0xCC, 0xCC, 0x76, 0x00,
0x00, 0xCC, 0x00, 0xCC, 0xCC, 0x7C, 0x0C, 0xF8,
0xC3, 0x18, 0x3C, 0x66, 0x66, 0x3C, 0x18, 0x00,
0xCC, 0x00, 0xCC, 0xCC, 0xCC, 0xCC, 0x78, 0x00,
0x18, 0x18, 0x7E, 0xC0, 0xC0, 0x7E, 0x18, 0x18,
0x38, 0x6C, 0x64, 0xF0, 0x60, 0xE6, 0xFC, 0x00,
0xCC, 0xCC, 0x78, 0x30, 0xFC, 0x30, 0xFC, 0x30,
0xF8, 0xCC, 0xCC, 0xFA, 0xC6, 0xCF, 0xC6, 0xC3,
0x0E, 0x1B, 0x18, 0x3C, 0x18, 0x18, 0xD8, 0x70,
0x1C, 0x00, 0x78, 0x0C, 0x7C, 0xCC, 0x76, 0x00,
0x38, 0x00, 0x70, 0x30, 0x30, 0x30, 0x78, 0x00,
0x00, 0x1C, 0x00, 0x78, 0xCC, 0xCC, 0x78, 0x00,
0x00, 0x1C, 0x00, 0xCC, 0xCC, 0xCC, 0x76, 0x00,
0x00, 0xF8, 0x00, 0xB8, 0xCC, 0xCC, 0xCC, 0x00,
0xFC, 0x00, 0xCC, 0xEC, 0xFC, 0xDC, 0xCC, 0x00,
0x3C, 0x6C, 0x6C, 0x3E, 0x00, 0x7E, 0x00, 0x00,
0x38, 0x6C, 0x6C, 0x38, 0x00, 0x7C, 0x00, 0x00,
0x18, 0x00, 0x18, 0x18, 0x30, 0x66, 0x3C, 0x00,
0x00, 0x00, 0x00, 0xFC, 0xC0, 0xC0, 0x00, 0x00,
0x00, 0x00, 0x00, 0xFC, 0x0C, 0x0C, 0x00, 0x00,
0xC6, 0xCC, 0xD8, 0x36, 0x6B, 0xC2, 0x84, 0x0F,
0xC3, 0xC6, 0xCC, 0xDB, 0x37, 0x6D, 0xCF, 0x03,
0x18, 0x00, 0x18, 0x18, 0x3C, 0x3C, 0x18, 0x00,
0x00, 0x33, 0x66, 0xCC, 0x66, 0x33, 0x00, 0x00,
0x00, 0xCC, 0x66, 0x33, 0x66, 0xCC, 0x00, 0x00,
0x22, 0x88, 0x22, 0x88, 0x22, 0x88, 0x22, 0x88,
0x55, 0xAA, 0x55, 0xAA, 0x55, 0xAA, 0x55, 0xAA,
0xDB, 0xF6, 0xDB, 0x6F, 0xDB, 0x7E, 0xD7, 0xED,
0x18, 0x18, 0x18, 0x18, 0x18, 0x18, 0x18, 0x18,
0x18, 0x18, 0x18, 0x18, 0xF8, 0x18, 0x18, 0x18,
0x18, 0x18, 0xF8, 0x18, 0xF8, 0x18, 0x18, 0x18,
0x36, 0x36, 0x36, 0x36, 0xF6, 0x36, 0x36, 0x36,
0x00, 0x00, 0x00, 0x00, 0xFE, 0x36, 0x36, 0x36,
0x00, 0x00, 0xF8, 0x18, 0xF8, 0x18, 0x18, 0x18,
0x36, 0x36, 0xF6, 0x06, 0xF6, 0x36, 0x36, 0x36,
0x36, 0x36, 0x36, 0x36, 0x36, 0x36, 0x36, 0x36,
0x00, 0x00, 0xFE, 0x06, 0xF6, 0x36, 0x36, 0x36,
0x36, 0x36, 0xF6, 0x06, 0xFE, 0x00, 0x00, 0x00,
0x36, 0x36, 0x36, 0x36, 0xFE, 0x00, 0x00, 0x00,
0x18, 0x18, 0xF8, 0x18, 0xF8, 0x00, 0x00, 0x00,
0x00, 0x00, 0x00, 0x00, 0xF8, 0x18, 0x18, 0x18,
0x18, 0x18, 0x18, 0x18, 0x1F, 0x00, 0x00, 0x00,
0x18, 0x18, 0x18, 0x18, 0xFF, 0x00, 0x00, 0x00,
0x00, 0x00, 0x00, 0x00, 0xFF, 0x18, 0x18, 0x18,
0x18, 0x18, 0x18, 0x18, 0x1F, 0x18, 0x18, 0x18,
```

```
        0x00, 0x00, 0x00, 0x00, 0xFF, 0x00, 0x00, 0x00,
        0x18, 0x18, 0x18, 0x18, 0xFF, 0x18, 0x18, 0x18,
        0x18, 0x18, 0x1F, 0x18, 0x1F, 0x18, 0x18, 0x18,
        0x36, 0x36, 0x36, 0x36, 0x37, 0x36, 0x36, 0x36,
        0x36, 0x36, 0x37, 0x30, 0x3F, 0x00, 0x00, 0x00,
        0x00, 0x00, 0x3F, 0x30, 0x37, 0x36, 0x36, 0x36,
        0x36, 0x36, 0xF7, 0x00, 0xFF, 0x00, 0x00, 0x00,
        0x00, 0x00, 0xFF, 0x00, 0xF7, 0x36, 0x36, 0x36,
        0x36, 0x36, 0x37, 0x30, 0x37, 0x36, 0x36, 0x36,
        0x00, 0x00, 0xFF, 0x00, 0xFF, 0x00, 0x00, 0x00,
        0x36, 0x36, 0xF7, 0x00, 0xF7, 0x36, 0x36, 0x36,
        0x18, 0x18, 0xFF, 0x00, 0xFF, 0x00, 0x00, 0x00,
        0x36, 0x36, 0x36, 0x36, 0xFF, 0x00, 0x00, 0x00,
        0x00, 0x00, 0xFF, 0x00, 0xFF, 0x18, 0x18, 0x18,
        0x00, 0x00, 0x00, 0x00, 0xFF, 0x36, 0x36, 0x36,
        0x36, 0x36, 0x36, 0x36, 0x3F, 0x00, 0x00, 0x00,
        0x18, 0x18, 0x1F, 0x18, 0x1F, 0x00, 0x00, 0x00,
        0x00, 0x00, 0x1F, 0x18, 0x1F, 0x18, 0x18, 0x18,
        0x00, 0x00, 0x00, 0x00, 0x3F, 0x36, 0x36, 0x36,
        0x36, 0x36, 0x36, 0x36, 0xFF, 0x36, 0x36, 0x36,
        0x18, 0x18, 0xFF, 0x18, 0xFF, 0x18, 0x18, 0x18,
        0x18, 0x18, 0x18, 0x18, 0xF8, 0x00, 0x00, 0x00,
        0x00, 0x00, 0x00, 0x00, 0x1F, 0x18, 0x18, 0x18,
        0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF,
        0x00, 0x00, 0x00, 0x00, 0xFF, 0xFF, 0xFF, 0xFF,
        0xF0, 0xF0, 0xF0, 0xF0, 0xF0, 0xF0, 0xF0, 0xF0,
        0x0F, 0x0F, 0x0F, 0x0F, 0x0F, 0x0F, 0x0F, 0x0F,
        0xFF, 0xFF, 0xFF, 0xFF, 0x00, 0x00, 0x00, 0x00,
        0x00, 0x00, 0x76, 0xDC, 0xC8, 0xDC, 0x76, 0x00,
        0x00, 0x78, 0xCC, 0xF8, 0xCC, 0xF8, 0xC0, 0xC0,
        0x00, 0xFC, 0xCC, 0xC0, 0xC0, 0xC0, 0xC0, 0x00,
        0x00, 0x00, 0xFE, 0x6C, 0x6C, 0x6C, 0x6C, 0x00,
        0xFC, 0xCC, 0x60, 0x30, 0x60, 0xCC, 0xFC, 0x00,
        0x00, 0x00, 0x7E, 0xD8, 0xD8, 0xD8, 0x70, 0x00,
        0x00, 0x66, 0x66, 0x66, 0x66, 0x7C, 0x60, 0xC0,
        0x00, 0x76, 0xDC, 0x18, 0x18, 0x18, 0x18, 0x00,
        0xFC, 0x30, 0x78, 0xCC, 0xCC, 0x78, 0x30, 0xFC,
        0x38, 0x6C, 0xC6, 0xFE, 0xC6, 0x6C, 0x38, 0x00,
        0x38, 0x6C, 0xC6, 0xC6, 0x6C, 0x6C, 0xEE, 0x00,
        0x1C, 0x30, 0x18, 0x7C, 0xCC, 0xCC, 0x78, 0x00,
        0x00, 0x00, 0x7E, 0xDB, 0xDB, 0x7E, 0x00, 0x00,
        0x06, 0x0C, 0x7E, 0xDB, 0xDB, 0x7E, 0x60, 0xC0,
        0x38, 0x60, 0xC0, 0xF8, 0xC0, 0x60, 0x38, 0x00,
        0x78, 0xCC, 0xCC, 0xCC, 0xCC, 0xCC, 0xCC, 0x00,
        0x00, 0x7E, 0x00, 0x7E, 0x00, 0x7E, 0x00, 0x00,
        0x18, 0x18, 0x7E, 0x18, 0x18, 0x00, 0x7E, 0x00,
        0x60, 0x30, 0x18, 0x30, 0x60, 0x00, 0xFC, 0x00,
        0x18, 0x30, 0x60, 0x30, 0x18, 0x00, 0xFC, 0x00,
        0x0E, 0x1B, 0x1B, 0x18, 0x18, 0x18, 0x18, 0x18,
        0x18, 0x18, 0x18, 0x18, 0x18, 0xD8, 0xD8, 0x70,
        0x18, 0x18, 0x00, 0x7E, 0x00, 0x18, 0x18, 0x00,
        0x00, 0x76, 0xDC, 0x00, 0x76, 0xDC, 0x00, 0x00,
        0x38, 0x6C, 0x6C, 0x38, 0x00, 0x00, 0x00, 0x00,
        0x00, 0x00, 0x00, 0x18, 0x18, 0x00, 0x00, 0x00,
        0x00, 0x00, 0x00, 0x00, 0x18, 0x00, 0x00, 0x00,
        0x0F, 0x0C, 0x0C, 0x0C, 0xEC, 0x6C, 0x3C, 0x1C,
        0x58, 0x6C, 0x6C, 0x6C, 0x6C, 0x00, 0x00, 0x00,
        0x70, 0x98, 0x30, 0x60, 0xF8, 0x00, 0x00, 0x00,
        0x00, 0x00, 0x3C, 0x3C, 0x3C, 0x3C, 0x00, 0x00,
        0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00
};
```

```
static unsigned char g_8x16_font[4096] =
{
    0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
    0x00, 0x00, 0x7E, 0x81, 0xA5, 0x81, 0x81, 0xBD, 0x99, 0x81, 0x81, 0x7E, 0x00, 0x00, 0x00, 0x00,
    0x00, 0x00, 0x7E, 0xFF, 0xDB, 0xFF, 0xFF, 0xC3, 0xE7, 0xFF, 0xFF, 0x7E, 0x00, 0x00, 0x00, 0x00,
    0x00, 0x00, 0x00, 0x00, 0x6C, 0xFE, 0xFE, 0xFE, 0xFE, 0x7C, 0x38, 0x10, 0x00, 0x00, 0x00, 0x00,
    0x00, 0x00, 0x00, 0x00, 0x10, 0x38, 0x7C, 0xFE, 0x7C, 0x38, 0x10, 0x00, 0x00, 0x00, 0x00, 0x00,
    0x00, 0x00, 0x00, 0x18, 0x3C, 0x3C, 0xE7, 0xE7, 0xE7, 0x99, 0x18, 0x3C, 0x00, 0x00, 0x00, 0x00,
    0x00, 0x00, 0x00, 0x18, 0x3C, 0x7E, 0xFF, 0xFF, 0x7E, 0x18, 0x18, 0x3C, 0x00, 0x00, 0x00, 0x00,
    0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x18, 0x3C, 0x3C, 0x18, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
    0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xE7, 0xC3, 0xC3, 0xE7, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF,
    0x00, 0x00, 0x00, 0x00, 0x00, 0x3C, 0x66, 0x42, 0x42, 0x66, 0x3C, 0x00, 0x00, 0x00, 0x00, 0x00,
    0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xC3, 0x99, 0xBD, 0xBD, 0x99, 0xC3, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF,
    0x00, 0x00, 0x1E, 0x0E, 0x1A, 0x32, 0x78, 0xCC, 0xCC, 0xCC, 0xCC, 0x78, 0x00, 0x00, 0x00, 0x00,
    0x00, 0x00, 0x3C, 0x66, 0x66, 0x66, 0x66, 0x3C, 0x18, 0x7E, 0x18, 0x18, 0x00, 0x00, 0x00, 0x00,
    0x00, 0x00, 0x3F, 0x33, 0x3F, 0x30, 0x30, 0x30, 0x30, 0x70, 0xF0, 0xE0, 0x00, 0x00, 0x00, 0x00,
    0x00, 0x00, 0x7F, 0x63, 0x7F, 0x63, 0x63, 0x63, 0x63, 0x67, 0xE7, 0xE6, 0xC0, 0x00, 0x00, 0x00,
    0x00, 0x00, 0x00, 0x18, 0x18, 0xDB, 0x3C, 0xE7, 0x3C, 0xDB, 0x18, 0x18, 0x00, 0x00, 0x00, 0x00,
    0x00, 0x80, 0xC0, 0xE0, 0xF0, 0xF8, 0xFE, 0xF8, 0xF0, 0xE0, 0xC0, 0x80, 0x00, 0x00, 0x00, 0x00,
    0x00, 0x02, 0x06, 0x0E, 0x1E, 0x3E, 0xFE, 0x3E, 0x1E, 0x0E, 0x06, 0x02, 0x00, 0x00, 0x00, 0x00,
    0x00, 0x00, 0x18, 0x3C, 0x7E, 0x18, 0x18, 0x18, 0x18, 0x7E, 0x3C, 0x18, 0x00, 0x00, 0x00, 0x00,
    0x00, 0x00, 0x66, 0x66, 0x66, 0x66, 0x66, 0x66, 0x66, 0x00, 0x66, 0x66, 0x00, 0x00, 0x00, 0x00,
    0x00, 0x00, 0x7F, 0xDB, 0xDB, 0xDB, 0x7B, 0x1B, 0x1B, 0x1B, 0x1B, 0x1B, 0x00, 0x00, 0x00, 0x00,
    0x00, 0x7C, 0xC6, 0x60, 0x38, 0x6C, 0xC6, 0xC6, 0x6C, 0x38, 0x0C, 0xC6, 0x7C, 0x00, 0x00, 0x00,
    0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0xFE, 0xFE, 0xFE, 0xFE, 0x00, 0x00, 0x00, 0x00,
    0x00, 0x00, 0x18, 0x3C, 0x7E, 0x18, 0x18, 0x18, 0x18, 0x7E, 0x3C, 0x18, 0x7E, 0x00, 0x00, 0x00,
    0x00, 0x00, 0x18, 0x3C, 0x7E, 0x18, 0x18, 0x18, 0x18, 0x18, 0x18, 0x18, 0x00, 0x00, 0x00, 0x00,
    0x00, 0x00, 0x18, 0x18, 0x18, 0x18, 0x18, 0x18, 0x18, 0x7E, 0x3C, 0x18, 0x00, 0x00, 0x00, 0x00,
    0x00, 0x00, 0x00, 0x00, 0x00, 0x18, 0x0C, 0xFE, 0x0C, 0x18, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
    0x00, 0x00, 0x00, 0x00, 0x00, 0x30, 0x60, 0xFE, 0x60, 0x30, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
    0x00, 0x00, 0x00, 0x00, 0x00, 0xC0, 0xC0, 0xC0, 0xC0, 0xFE, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
    0x00, 0x00, 0x00, 0x00, 0x00, 0x28, 0x6C, 0xFE, 0x6C, 0x28, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
    0x00, 0x00, 0x00, 0x00, 0x10, 0x38, 0x38, 0x7C, 0x7C, 0xFE, 0xFE, 0x00, 0x00, 0x00, 0x00, 0x00,
    0x00, 0x00, 0x00, 0x00, 0xFE, 0xFE, 0x7C, 0x7C, 0x38, 0x38, 0x10, 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, 0x00,
    0x00, 0x66, 0x66, 0x66, 0x24, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
    0x00, 0x00, 0x00, 0x6C, 0x6C, 0xFE, 0x6C, 0x6C, 0x6C, 0xFE, 0x6C, 0x6C, 0x00, 0x00, 0x00, 0x00,
    0x18, 0x18, 0x7C, 0xC6, 0xC2, 0xC0, 0x7C, 0x06, 0x86, 0xC6, 0x7C, 0x18, 0x18, 0x00, 0x00, 0x00,
    0x00, 0x00, 0x00, 0x00, 0xC2, 0xC6, 0x0C, 0x18, 0x30, 0x60, 0xC6, 0x86, 0x00, 0x00, 0x00, 0x00,
    0x00, 0x00, 0x38, 0x6C, 0x6C, 0x38, 0x76, 0xDC, 0xCC, 0xCC, 0xCC, 0x76, 0x00, 0x00, 0x00, 0x00,
    0x00, 0x30, 0x30, 0x30, 0x60, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
    0x00, 0x00, 0x0C, 0x18, 0x30, 0x30, 0x30, 0x30, 0x30, 0x30, 0x18, 0x0C, 0x00, 0x00, 0x00, 0x00,
    0x00, 0x00, 0x30, 0x18, 0x0C, 0x0C, 0x0C, 0x0C, 0x0C, 0x0C, 0x18, 0x30, 0x00, 0x00, 0x00, 0x00,
    0x00, 0x00, 0x00, 0x00, 0x00, 0x66, 0x3C, 0xFF, 0x3C, 0x66, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
    0x00, 0x00, 0x00, 0x00, 0x00, 0x18, 0x18, 0x7E, 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, 0x00, 0xFE, 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, 0x00, 0x00, 0x02, 0x06, 0x0C, 0x18, 0x30, 0x60, 0xC0, 0x80, 0x00, 0x00, 0x00, 0x00,
    0x00, 0x00, 0x7C, 0xC6, 0xC6, 0xCE, 0xD6, 0xD6, 0xE6, 0xC6, 0xC6, 0x7C, 0x00, 0x00, 0x00, 0x00,
    0x00, 0x00, 0x18, 0x38, 0x78, 0x18, 0x18, 0x18, 0x18, 0x18, 0x18, 0x7E, 0x00, 0x00, 0x00, 0x00,
    0x00, 0x00, 0x7C, 0xC6, 0x06, 0x0C, 0x18, 0x30, 0x60, 0xC0, 0xC6, 0xFE, 0x00, 0x00, 0x00, 0x00,
    0x00, 0x00, 0x7C, 0xC6, 0x06, 0x06, 0x3C, 0x06, 0x06, 0x06, 0xC6, 0x7C, 0x00, 0x00, 0x00, 0x00,
    0x00, 0x00, 0x0C, 0x1C, 0x3C, 0x6C, 0xCC, 0xFE, 0x0C, 0x0C, 0x0C, 0x1E, 0x00, 0x00, 0x00, 0x00,
    0x00, 0x00, 0xFE, 0xC0, 0xC0, 0xC0, 0xFC, 0x0E, 0x06, 0x06, 0xC6, 0x7C, 0x00, 0x00, 0x00, 0x00,
    0x00, 0x00, 0x38, 0x60, 0xC0, 0xC0, 0xFC, 0xC6, 0xC6, 0xC6, 0xC6, 0x7C, 0x00, 0x00, 0x00, 0x00,
    0x00, 0x00, 0xFE, 0xC6, 0x06, 0x06, 0x0C, 0x18, 0x30, 0x30, 0x30, 0x30, 0x00, 0x00, 0x00, 0x00,
    0x00, 0x00, 0x7C, 0xC6, 0xC6, 0xC6, 0x7C, 0xC6, 0xC6, 0xC6, 0xC6, 0x7C, 0x00, 0x00, 0x00, 0x00,
    0x00, 0x00, 0x7C, 0xC6, 0xC6, 0xC6, 0x7E, 0x06, 0x06, 0x06, 0x0C, 0x78, 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, 0x30, 0x00, 0x00, 0x00,
    0x00, 0x00, 0x00, 0x06, 0x0C, 0x18, 0x30, 0x60, 0x30, 0x18, 0x0C, 0x06, 0x00, 0x00, 0x00, 0x00,
    0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0xFE, 0x00, 0x00, 0xFE, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
    0x00, 0x00, 0x00, 0x60, 0x30, 0x18, 0x0C, 0x06, 0x0C, 0x18, 0x30, 0x60, 0x00, 0x00, 0x00, 0x00,
    0x00, 0x00, 0x7C, 0xC6, 0xC6, 0x0C, 0x18, 0x18, 0x18, 0x00, 0x18, 0x18, 0x00, 0x00, 0x00, 0x00,
    0x00, 0x00, 0x00, 0x7C, 0xC6, 0xC6, 0xDE, 0xDE, 0xDE, 0xDC, 0xC0, 0x7C, 0x00, 0x00, 0x00, 0x00,
    0x00, 0x00, 0x10, 0x38, 0x6C, 0xC6, 0xC6, 0xFE, 0xC6, 0xC6, 0xC6, 0xC6, 0x00, 0x00, 0x00, 0x00,
    0x00, 0x00, 0xFC, 0x66, 0x66, 0x66, 0x7C, 0x66, 0x66, 0x66, 0x66, 0xFC, 0x00, 0x00, 0x00, 0x00,
    0x00, 0x00, 0x3C, 0x66, 0xC2, 0xC0, 0xC0, 0xC0, 0xC0, 0xC2, 0x66, 0x3C, 0x00, 0x00, 0x00, 0x00,
    0x00, 0x00, 0xF8, 0x6C, 0x66, 0x66, 0x66, 0x66, 0x66, 0x66, 0x6C, 0xF8, 0x00, 0x00, 0x00, 0x00,
    0x00, 0x00, 0xFE, 0x66, 0x62, 0x68, 0x78, 0x68, 0x60, 0x62, 0x66, 0xFE, 0x00, 0x00, 0x00, 0x00,
    0x00, 0x00, 0xFE, 0x66, 0x62, 0x68, 0x78, 0x68, 0x60, 0x60, 0x60, 0xF0, 0x00, 0x00, 0x00, 0x00,
    0x00, 0x00, 0x3C, 0x66, 0xC2, 0xC0, 0xC0, 0xDE, 0xC6, 0xC6, 0x66, 0x3A, 0x00, 0x00, 0x00, 0x00,
    0x00, 0x00, 0xC6, 0xC6, 0xC6, 0xC6, 0xFE, 0xC6, 0xC6, 0xC6, 0xC6, 0xC6, 0x00, 0x00, 0x00, 0x00,
    0x00, 0x00, 0x3C, 0x18, 0x18, 0x18, 0x18, 0x18, 0x18, 0x18, 0x18, 0x3C, 0x00, 0x00, 0x00, 0x00,
    0x00, 0x00, 0x1E, 0x0C, 0x0C, 0x0C, 0x0C, 0x0C, 0xCC, 0xCC, 0xCC, 0x78, 0x00, 0x00, 0x00, 0x00,
```

```
0x00, 0x00, 0xE6, 0x66, 0x6C, 0x6C, 0x78, 0x78, 0x6C, 0x66, 0x66, 0xE6, 0x00, 0x00, 0x00, 0x00,
0x00, 0x00, 0xF0, 0x60, 0x60, 0x60, 0x60, 0x60, 0x60, 0x62, 0x66, 0xFE, 0x00, 0x00, 0x00, 0x00,
0x00, 0x00, 0xC6, 0xEE, 0xFE, 0xFE, 0xD6, 0xC6, 0xC6, 0xC6, 0xC6, 0xC6, 0x00, 0x00, 0x00, 0x00,
0x00, 0x00, 0xC6, 0xE6, 0xF6, 0xFE, 0xDE, 0xCE, 0xC6, 0xC6, 0xC6, 0xC6, 0x00, 0x00, 0x00, 0x00,
0x00, 0x00, 0x38, 0x6C, 0xC6, 0xC6, 0xC6, 0xC6, 0xC6, 0xC6, 0x6C, 0x38, 0x00, 0x00, 0x00, 0x00,
0x00, 0x00, 0xFC, 0x66, 0x66, 0x66, 0x7C, 0x60, 0x60, 0x60, 0x60, 0xF0, 0x00, 0x00, 0x00, 0x00,
0x00, 0x00, 0x7C, 0xC6, 0xC6, 0xC6, 0xC6, 0xC6, 0xC6, 0xD6, 0xDE, 0x7C, 0x0C, 0x0E, 0x00, 0x00,
0x00, 0x00, 0xFC, 0x66, 0x66, 0x66, 0x7C, 0x6C, 0x66, 0x66, 0x66, 0xE6, 0x00, 0x00, 0x00, 0x00,
0x00, 0x00, 0x7C, 0xC6, 0xC6, 0x60, 0x38, 0x0C, 0x06, 0xC6, 0xC6, 0x7C, 0x00, 0x00, 0x00, 0x00,
0x00, 0x00, 0x7E, 0x7E, 0x5A, 0x18, 0x18, 0x18, 0x18, 0x18, 0x18, 0x3C, 0x00, 0x00, 0x00, 0x00,
0x00, 0x00, 0xC6, 0xC6, 0xC6, 0xC6, 0xC6, 0xC6, 0xC6, 0xC6, 0xC6, 0x7C, 0x00, 0x00, 0x00, 0x00,
0x00, 0x00, 0xC6, 0xC6, 0xC6, 0xC6, 0xC6, 0xC6, 0xC6, 0x6C, 0x38, 0x10, 0x00, 0x00, 0x00, 0x00,
0x00, 0x00, 0xC6, 0xC6, 0xC6, 0xC6, 0xC6, 0xD6, 0xD6, 0xFE, 0x6C, 0x6C, 0x00, 0x00, 0x00, 0x00,
0x00, 0x00, 0xC6, 0xC6, 0x6C, 0x6C, 0x38, 0x38, 0x6C, 0x6C, 0xC6, 0xC6, 0x00, 0x00, 0x00, 0x00,
0x00, 0x00, 0x66, 0x66, 0x66, 0x66, 0x3C, 0x18, 0x18, 0x18, 0x18, 0x3C, 0x00, 0x00, 0x00, 0x00,
0x00, 0x00, 0xFE, 0xC6, 0x86, 0x0C, 0x18, 0x30, 0x60, 0xC2, 0xC6, 0xFE, 0x00, 0x00, 0x00, 0x00,
0x00, 0x00, 0x3C, 0x30, 0x30, 0x30, 0x30, 0x30, 0x30, 0x30, 0x30, 0x3C, 0x00, 0x00, 0x00, 0x00,
0x00, 0x00, 0x00, 0x80, 0xC0, 0xE0, 0x70, 0x38, 0x1C, 0x0E, 0x06, 0x02, 0x00, 0x00, 0x00, 0x00,
0x00, 0x00, 0x3C, 0x0C, 0x0C, 0x0C, 0x0C, 0x0C, 0x0C, 0x0C, 0x0C, 0x3C, 0x00, 0x00, 0x00, 0x00,
0x10, 0x38, 0x6C, 0xC6, 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, 0xFF, 0x00, 0x00,
0x30, 0x30, 0x18, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
0x00, 0x00, 0x00, 0x00, 0x00, 0x78, 0x0C, 0x7C, 0xCC, 0xCC, 0xCC, 0x76, 0x00, 0x00, 0x00, 0x00,
0x00, 0x00, 0xE0, 0x60, 0x60, 0x78, 0x6C, 0x66, 0x66, 0x66, 0x66, 0xDC, 0x00, 0x00, 0x00, 0x00,
0x00, 0x00, 0x00, 0x00, 0x00, 0x7C, 0xC6, 0xC0, 0xC0, 0xC0, 0xC6, 0x7C, 0x00, 0x00, 0x00, 0x00,
0x00, 0x00, 0x1C, 0x0C, 0x0C, 0x3C, 0x6C, 0xCC, 0xCC, 0xCC, 0xCC, 0x76, 0x00, 0x00, 0x00, 0x00,
0x00, 0x00, 0x00, 0x00, 0x00, 0x7C, 0xC6, 0xFE, 0xC0, 0xC0, 0xC6, 0x7C, 0x00, 0x00, 0x00, 0x00,
0x00, 0x00, 0x38, 0x6C, 0x64, 0x60, 0xF0, 0x60, 0x60, 0x60, 0x60, 0xF0, 0x00, 0x00, 0x00, 0x00,
0x00, 0x00, 0x00, 0x00, 0x00, 0x76, 0xCC, 0xCC, 0xCC, 0xCC, 0xCC, 0x7C, 0x0C, 0xCC, 0x78, 0x00,
0x00, 0x00, 0xE0, 0x60, 0x60, 0x6C, 0x76, 0x66, 0x66, 0x66, 0x66, 0xE6, 0x00, 0x00, 0x00, 0x00,
0x00, 0x00, 0x18, 0x18, 0x00, 0x38, 0x18, 0x18, 0x18, 0x18, 0x18, 0x3C, 0x00, 0x00, 0x00, 0x00,
0x00, 0x00, 0x06, 0x06, 0x00, 0x0E, 0x06, 0x06, 0x06, 0x06, 0x06, 0x06, 0x66, 0x66, 0x3C, 0x00,
0x00, 0x00, 0xE0, 0x60, 0x60, 0x66, 0x6C, 0x78, 0x78, 0x6C, 0x66, 0xE6, 0x00, 0x00, 0x00, 0x00,
0x00, 0x00, 0x38, 0x18, 0x18, 0x18, 0x18, 0x18, 0x18, 0x18, 0x18, 0x3C, 0x00, 0x00, 0x00, 0x00,
0x00, 0x00, 0x00, 0x00, 0x00, 0xEC, 0xFE, 0xD6, 0xD6, 0xD6, 0xD6, 0xD6, 0x00, 0x00, 0x00, 0x00,
0x00, 0x00, 0x00, 0x00, 0x00, 0xDC, 0x66, 0x66, 0x66, 0x66, 0x66, 0x66, 0x00, 0x00, 0x00, 0x00,
0x00, 0x00, 0x00, 0x00, 0x00, 0x7C, 0xC6, 0xC6, 0xC6, 0xC6, 0xC6, 0x7C, 0x00, 0x00, 0x00, 0x00,
0x00, 0x00, 0x00, 0x00, 0x00, 0xDC, 0x66, 0x66, 0x66, 0x66, 0x66, 0x7C, 0x60, 0x60, 0xF0, 0x00,
0x00, 0x00, 0x00, 0x00, 0x00, 0x76, 0xCC, 0xCC, 0xCC, 0xCC, 0xCC, 0x7C, 0x0C, 0x0C, 0x1E, 0x00,
0x00, 0x00, 0x00, 0x00, 0x00, 0xDC, 0x76, 0x62, 0x60, 0x60, 0x60, 0xF0, 0x00, 0x00, 0x00, 0x00,
0x00, 0x00, 0x00, 0x00, 0x00, 0x7C, 0xC6, 0x60, 0x38, 0x0C, 0xC6, 0x7C, 0x00, 0x00, 0x00, 0x00,
0x00, 0x00, 0x10, 0x30, 0x30, 0xFC, 0x30, 0x30, 0x30, 0x30, 0x36, 0x1C, 0x00, 0x00, 0x00, 0x00,
0x00, 0x00, 0x00, 0x00, 0x00, 0xCC, 0xCC, 0xCC, 0xCC, 0xCC, 0xCC, 0x76, 0x00, 0x00, 0x00, 0x00,
0x00, 0x00, 0x00, 0x00, 0x00, 0x66, 0x66, 0x66, 0x66, 0x66, 0x3C, 0x18, 0x00, 0x00, 0x00, 0x00,
0x00, 0x00, 0x00, 0x00, 0x00, 0xC6, 0xC6, 0xC6, 0xD6, 0xD6, 0xFE, 0x6C, 0x00, 0x00, 0x00, 0x00,
0x00, 0x00, 0x00, 0x00, 0x00, 0xC6, 0x6C, 0x38, 0x38, 0x38, 0x6C, 0xC6, 0x00, 0x00, 0x00, 0x00,
0x00, 0x00, 0x00, 0x00, 0x00, 0xC6, 0xC6, 0xC6, 0xC6, 0xC6, 0xC6, 0x7E, 0x06, 0x0C, 0xF8, 0x00,
0x00, 0x00, 0x00, 0x00, 0x00, 0xFE, 0xCC, 0x18, 0x30, 0x60, 0xC6, 0xFE, 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, 0x00, 0x18, 0x18, 0x18, 0x18, 0x18, 0x00, 0x00, 0x00, 0x00,
0x00, 0x00, 0x70, 0x18, 0x18, 0x18, 0x0E, 0x18, 0x18, 0x18, 0x18, 0x70, 0x00, 0x00, 0x00, 0x00,
0x00, 0x00, 0x76, 0xDC, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
0x00, 0x00, 0x00, 0x00, 0x10, 0x38, 0x6C, 0xC6, 0xC6, 0xC6, 0xFE, 0x00, 0x00, 0x00, 0x00, 0x00,
0x00, 0x00, 0x3C, 0x66, 0xC2, 0xC0, 0xC0, 0xC0, 0xC2, 0x66, 0x3C, 0x0C, 0x06, 0x7C, 0x00, 0x00,
0x00, 0x00, 0xCC, 0xCC, 0x00, 0xCC, 0xCC, 0xCC, 0xCC, 0xCC, 0xCC, 0x76, 0x00, 0x00, 0x00, 0x00,
0x00, 0x0C, 0x18, 0x30, 0x00, 0x7C, 0xC6, 0xFE, 0xC0, 0xC0, 0xC6, 0x7C, 0x00, 0x00, 0x00, 0x00,
0x00, 0x10, 0x38, 0x6C, 0x00, 0x78, 0x0C, 0x7C, 0xCC, 0xCC, 0xCC, 0x76, 0x00, 0x00, 0x00, 0x00,
0x00, 0x00, 0xCC, 0xCC, 0x00, 0x78, 0x0C, 0x7C, 0xCC, 0xCC, 0xCC, 0x76, 0x00, 0x00, 0x00, 0x00,
0x00, 0x60, 0x30, 0x18, 0x00, 0x78, 0x0C, 0x7C, 0xCC, 0xCC, 0xCC, 0x76, 0x00, 0x00, 0x00, 0x00,
0x00, 0x38, 0x6C, 0x38, 0x00, 0x78, 0x0C, 0x7C, 0xCC, 0xCC, 0xCC, 0x76, 0x00, 0x00, 0x00, 0x00,
0x00, 0x00, 0x00, 0x00, 0x3C, 0x66, 0x60, 0x60, 0x66, 0x3C, 0x0C, 0x06, 0x3C, 0x00, 0x00, 0x00,
0x00, 0x10, 0x38, 0x6C, 0x00, 0x7C, 0xC6, 0xFE, 0xC0, 0xC0, 0xC6, 0x7C, 0x00, 0x00, 0x00, 0x00,
0x00, 0x00, 0xC6, 0xC6, 0x00, 0x7C, 0xC6, 0xFE, 0xC0, 0xC0, 0xC6, 0x7C, 0x00, 0x00, 0x00, 0x00,
0x00, 0x60, 0x30, 0x18, 0x00, 0x7C, 0xC6, 0xFE, 0xC0, 0xC0, 0xC6, 0x7C, 0x00, 0x00, 0x00, 0x00,
0x00, 0x00, 0x66, 0x66, 0x00, 0x38, 0x18, 0x18, 0x18, 0x18, 0x18, 0x3C, 0x00, 0x00, 0x00, 0x00,
0x00, 0x18, 0x3C, 0x66, 0x00, 0x38, 0x18, 0x18, 0x18, 0x18, 0x18, 0x3C, 0x00, 0x00, 0x00, 0x00,
0x00, 0x60, 0x30, 0x18, 0x00, 0x38, 0x18, 0x18, 0x18, 0x18, 0x18, 0x3C, 0x00, 0x00, 0x00, 0x00,
0x00, 0xC6, 0xC6, 0x10, 0x38, 0x6C, 0xC6, 0xC6, 0xFE, 0xC6, 0xC6, 0xC6, 0x00, 0x00, 0x00, 0x00,
0x38, 0x6C, 0x38, 0x00, 0x38, 0x6C, 0xC6, 0xC6, 0xFE, 0xC6, 0xC6, 0xC6, 0x00, 0x00, 0x00, 0x00,
0x18, 0x30, 0x60, 0x00, 0xFE, 0x66, 0x60, 0x7C, 0x60, 0x60, 0x66, 0xFE, 0x00, 0x00, 0x00, 0x00,
0x00, 0x00, 0x00, 0x00, 0x00, 0xCC, 0x76, 0x36, 0x7E, 0xD8, 0xD8, 0x6E, 0x00, 0x00, 0x00, 0x00,
0x00, 0x00, 0x3E, 0x6C, 0xCC, 0xCC, 0xFE, 0xCC, 0xCC, 0xCC, 0xCC, 0xCE, 0x00, 0x00, 0x00, 0x00,
0x00, 0x10, 0x38, 0x6C, 0x00, 0x7C, 0xC6, 0xC6, 0xC6, 0xC6, 0xC6, 0x7C, 0x00, 0x00, 0x00, 0x00,
0x00, 0x00, 0xC6, 0xC6, 0x00, 0x7C, 0xC6, 0xC6, 0xC6, 0xC6, 0xC6, 0x7C, 0x00, 0x00, 0x00, 0x00,
0x00, 0x60, 0x30, 0x18, 0x00, 0x7C, 0xC6, 0xC6, 0xC6, 0xC6, 0xC6, 0x7C, 0x00, 0x00, 0x00, 0x00,
0x00, 0x30, 0x78, 0xCC, 0x00, 0xCC, 0xCC, 0xCC, 0xCC, 0xCC, 0xCC, 0x76, 0x00, 0x00, 0x00, 0x00,
0x00, 0x60, 0x30, 0x18, 0x00, 0xCC, 0xCC, 0xCC, 0xCC, 0xCC, 0xCC, 0x76, 0x00, 0x00, 0x00, 0x00,
```

```
0x00, 0x00, 0xC6, 0xC6, 0x00, 0xC6, 0xC6, 0xC6, 0xC6, 0xC6, 0xC6, 0x7E, 0x06, 0x0C, 0x78, 0x00,
0x00, 0xC6, 0xC6, 0x00, 0x38, 0x6C, 0xC6, 0xC6, 0xC6, 0xC6, 0x6C, 0x38, 0x00, 0x00, 0x00, 0x00,
0x00, 0xC6, 0xC6, 0x00, 0xC6, 0xC6, 0xC6, 0xC6, 0xC6, 0xC6, 0xC6, 0x7C, 0x00, 0x00, 0x00, 0x00,
0x00, 0x18, 0x18, 0x3C, 0x66, 0x60, 0x60, 0x60, 0x66, 0x3C, 0x18, 0x18, 0x00, 0x00, 0x00, 0x00,
0x00, 0x38, 0x6C, 0x64, 0x60, 0xF0, 0x60, 0x60, 0x60, 0x60, 0xE6, 0xFC, 0x00, 0x00, 0x00, 0x00,
0x00, 0x00, 0x66, 0x66, 0x3C, 0x18, 0x7E, 0x18, 0x7E, 0x18, 0x18, 0x18, 0x00, 0x00, 0x00, 0x00,
0x00, 0xF8, 0xCC, 0xCC, 0xF8, 0xC4, 0xCC, 0xDE, 0xCC, 0xCC, 0xCC, 0xC6, 0x00, 0x00, 0x00, 0x00,
0x00, 0x0E, 0x1B, 0x18, 0x18, 0x18, 0x7E, 0x18, 0x18, 0x18, 0x18, 0x18, 0xD8, 0x70, 0x00, 0x00,
0x00, 0x18, 0x30, 0x60, 0x00, 0x78, 0x0C, 0x7C, 0xCC, 0xCC, 0xCC, 0x76, 0x00, 0x00, 0x00, 0x00,
0x00, 0x0C, 0x18, 0x30, 0x00, 0x38, 0x18, 0x18, 0x18, 0x18, 0x18, 0x3C, 0x00, 0x00, 0x00, 0x00,
0x00, 0x18, 0x30, 0x60, 0x00, 0x7C, 0xC6, 0xC6, 0xC6, 0xC6, 0xC6, 0x7C, 0x00, 0x00, 0x00, 0x00,
0x00, 0x18, 0x30, 0x60, 0x00, 0xCC, 0xCC, 0xCC, 0xCC, 0xCC, 0xCC, 0x76, 0x00, 0x00, 0x00, 0x00,
0x00, 0x00, 0x76, 0xDC, 0x00, 0xDC, 0x66, 0x66, 0x66, 0x66, 0x66, 0x66, 0x00, 0x00, 0x00, 0x00,
0x76, 0xDC, 0x00, 0xC6, 0xE6, 0xF6, 0xFE, 0xDE, 0xCE, 0xC6, 0xC6, 0xC6, 0x00, 0x00, 0x00, 0x00,
0x00, 0x3C, 0x6C, 0x6C, 0x3E, 0x00, 0x7E, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
0x00, 0x38, 0x6C, 0x6C, 0x38, 0x00, 0x7C, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
0x00, 0x00, 0x30, 0x30, 0x00, 0x30, 0x30, 0x60, 0xC0, 0xC6, 0xC6, 0x7C, 0x00, 0x00, 0x00, 0x00,
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0xFE, 0xC0, 0xC0, 0xC0, 0xC0, 0x00, 0x00, 0x00, 0x00, 0x00,
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0xFE, 0x06, 0x06, 0x06, 0x06, 0x00, 0x00, 0x00, 0x00, 0x00,
0x00, 0xC0, 0xC0, 0xC2, 0xC6, 0xCC, 0x18, 0x30, 0x60, 0xCE, 0x93, 0x06, 0x0C, 0x1F, 0x00, 0x00,
0x00, 0xC0, 0xC0, 0xC2, 0xC6, 0xCC, 0x18, 0x30, 0x66, 0xCE, 0x9A, 0x3F, 0x06, 0x0F, 0x00, 0x00,
0x00, 0x00, 0x18, 0x18, 0x00, 0x18, 0x18, 0x18, 0x3C, 0x3C, 0x3C, 0x18, 0x00, 0x00, 0x00, 0x00,
0x00, 0x00, 0x00, 0x00, 0x00, 0x33, 0x66, 0xCC, 0x66, 0x33, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
0x00, 0x00, 0x00, 0x00, 0x00, 0xCC, 0x66, 0x33, 0x66, 0xCC, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
0x11, 0x44, 0x11, 0x44, 0x11, 0x44, 0x11, 0x44, 0x11, 0x44, 0x11, 0x44, 0x11, 0x44, 0x11, 0x44,
0x55, 0xAA, 0x55, 0xAA, 0x55, 0xAA, 0x55, 0xAA, 0x55, 0xAA, 0x55, 0xAA, 0x55, 0xAA, 0x55, 0xAA,
0xDD, 0x77, 0xDD, 0x77, 0xDD, 0x77, 0xDD, 0x77, 0xDD, 0x77, 0xDD, 0x77, 0xDD, 0x77, 0xDD, 0x77,
0x18, 0x18, 0x18, 0x18, 0x18, 0x18, 0x18, 0x18, 0x18, 0x18, 0x18, 0x18, 0x18, 0x18, 0x18, 0x18,
0x18, 0x18, 0x18, 0x18, 0x18, 0x18, 0x18, 0xF8, 0x18, 0x18, 0x18, 0x18, 0x18, 0x18, 0x18, 0x18,
0x18, 0x18, 0x18, 0x18, 0x18, 0xF8, 0x18, 0xF8, 0x18, 0x18, 0x18, 0x18, 0x18, 0x18, 0x18, 0x18,
0x36, 0x36, 0x36, 0x36, 0x36, 0x36, 0x36, 0xF6, 0x36, 0x36, 0x36, 0x36, 0x36, 0x36, 0x36, 0x36,
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0xFE, 0x36, 0x36, 0x36, 0x36, 0x36, 0x36, 0x36, 0x36,
0x00, 0x00, 0x00, 0x00, 0x00, 0xF8, 0x18, 0xF8, 0x18, 0x18, 0x18, 0x18, 0x18, 0x18, 0x18, 0x18,
0x36, 0x36, 0x36, 0x36, 0x36, 0xF6, 0x06, 0xF6, 0x36, 0x36, 0x36, 0x36, 0x36, 0x36, 0x36, 0x36,
0x36, 0x36, 0x36, 0x36, 0x36, 0x36, 0x36, 0x36, 0x36, 0x36, 0x36, 0x36, 0x36, 0x36, 0x36, 0x36,
0x00, 0x00, 0x00, 0x00, 0x00, 0xFE, 0x06, 0xF6, 0x36, 0x36, 0x36, 0x36, 0x36, 0x36, 0x36, 0x36,
0x36, 0x36, 0x36, 0x36, 0x36, 0xF6, 0x06, 0xFE, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
0x36, 0x36, 0x36, 0x36, 0x36, 0x36, 0x36, 0xFE, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
0x18, 0x18, 0x18, 0x18, 0x18, 0xF8, 0x18, 0xF8, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0xF8, 0x18, 0x18, 0x18, 0x18, 0x18, 0x18, 0x18, 0x18,
0x18, 0x18, 0x18, 0x18, 0x18, 0x18, 0x18, 0x1F, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
0x18, 0x18, 0x18, 0x18, 0x18, 0x18, 0x18, 0xFF, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0xFF, 0x18, 0x18, 0x18, 0x18, 0x18, 0x18, 0x18, 0x18,
0x18, 0x18, 0x18, 0x18, 0x18, 0x18, 0x18, 0x1F, 0x18, 0x18, 0x18, 0x18, 0x18, 0x18, 0x18, 0x18,
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0xFF, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
0x18, 0x18, 0x18, 0x18, 0x18, 0x18, 0x18, 0xFF, 0x18, 0x18, 0x18, 0x18, 0x18, 0x18, 0x18, 0x18,
0x18, 0x18, 0x18, 0x18, 0x18, 0x1F, 0x18, 0x1F, 0x18, 0x18, 0x18, 0x18, 0x18, 0x18, 0x18, 0x18,
0x36, 0x36, 0x36, 0x36, 0x36, 0x36, 0x36, 0x37, 0x36, 0x36, 0x36, 0x36, 0x36, 0x36, 0x36, 0x36,
0x36, 0x36, 0x36, 0x36, 0x36, 0x37, 0x30, 0x3F, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
0x00, 0x00, 0x00, 0x00, 0x00, 0x3F, 0x30, 0x37, 0x36, 0x36, 0x36, 0x36, 0x36, 0x36, 0x36, 0x36,
0x36, 0x36, 0x36, 0x36, 0x36, 0xF7, 0x00, 0xFF, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
0x00, 0x00, 0x00, 0x00, 0x00, 0xFF, 0x00, 0xF7, 0x36, 0x36, 0x36, 0x36, 0x36, 0x36, 0x36, 0x36,
0x36, 0x36, 0x36, 0x36, 0x36, 0x37, 0x30, 0x37, 0x36, 0x36, 0x36, 0x36, 0x36, 0x36, 0x36, 0x36,
0x00, 0x00, 0x00, 0x00, 0x00, 0xFF, 0x00, 0xFF, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
0x36, 0x36, 0x36, 0x36, 0x36, 0xF7, 0x00, 0xF7, 0x36, 0x36, 0x36, 0x36, 0x36, 0x36, 0x36, 0x36,
0x18, 0x18, 0x18, 0x18, 0x18, 0xFF, 0x00, 0xFF, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
0x36, 0x36, 0x36, 0x36, 0x36, 0x36, 0x36, 0xFF, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
0x00, 0x00, 0x00, 0x00, 0x00, 0xFF, 0x00, 0xFF, 0x18, 0x18, 0x18, 0x18, 0x18, 0x18, 0x18, 0x18,
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0xFF, 0x36, 0x36, 0x36, 0x36, 0x36, 0x36, 0x36, 0x36,
0x36, 0x36, 0x36, 0x36, 0x36, 0x36, 0x36, 0x3F, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
0x18, 0x18, 0x18, 0x18, 0x18, 0x1F, 0x18, 0x1F, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
0x00, 0x00, 0x00, 0x00, 0x00, 0x1F, 0x18, 0x1F, 0x18, 0x18, 0x18, 0x18, 0x18, 0x18, 0x18, 0x18,
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x3F, 0x36, 0x36, 0x36, 0x36, 0x36, 0x36, 0x36, 0x36,
0x36, 0x36, 0x36, 0x36, 0x36, 0x36, 0x36, 0xFF, 0x36, 0x36, 0x36, 0x36, 0x36, 0x36, 0x36, 0x36,
0x18, 0x18, 0x18, 0x18, 0x18, 0xFF, 0x18, 0xFF, 0x18, 0x18, 0x18, 0x18, 0x18, 0x18, 0x18, 0x18,
0x18, 0x18, 0x18, 0x18, 0x18, 0x18, 0x18, 0xF8, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x1F, 0x18, 0x18, 0x18, 0x18, 0x18, 0x18, 0x18, 0x18,
0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF,
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF,
0xF0, 0xF0, 0xF0, 0xF0, 0xF0, 0xF0, 0xF0, 0xF0, 0xF0, 0xF0, 0xF0, 0xF0, 0xF0, 0xF0, 0xF0, 0xF0,
0x0F, 0x0F, 0x0F, 0x0F, 0x0F, 0x0F, 0x0F, 0x0F, 0x0F, 0x0F, 0x0F, 0x0F, 0x0F, 0x0F, 0x0F, 0x0F,
0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
0x00, 0x00, 0x00, 0x00, 0x00, 0x76, 0xDC, 0xD8, 0xD8, 0xD8, 0xDC, 0x76, 0x00, 0x00, 0x00, 0x00,
0x00, 0x00, 0x00, 0x00, 0x00, 0xFC, 0xC6, 0xFC, 0xC6, 0xC6, 0xFC, 0xC0, 0xC0, 0xC0, 0x00, 0x00,
0x00, 0x00, 0xFE, 0xC6, 0xC6, 0xC0, 0xC0, 0xC0, 0xC0, 0xC0, 0xC0, 0xC0, 0x00, 0x00, 0x00, 0x00,
0x00, 0x00, 0x00, 0x00, 0x80, 0xFE, 0x6C, 0x6C, 0x6C, 0x6C, 0x6C, 0x6C, 0x00, 0x00, 0x00, 0x00,
0x00, 0x00, 0x00, 0xFE, 0xC6, 0x60, 0x30, 0x18, 0x30, 0x60, 0xC6, 0xFE, 0x00, 0x00, 0x00, 0x00,
```

```
    0x00, 0x00, 0x00, 0x00, 0x00, 0x7E, 0xD8, 0xD8, 0xD8, 0xD8, 0xD8, 0x70, 0x00, 0x00, 0x00, 0x00,
    0x00, 0x00, 0x00, 0x00, 0x66, 0x66, 0x66, 0x66, 0x66, 0x7C, 0x60, 0x60, 0xC0, 0x00, 0x00, 0x00,
    0x00, 0x00, 0x00, 0x00, 0x76, 0xDC, 0x18, 0x18, 0x18, 0x18, 0x18, 0x18, 0x00, 0x00, 0x00, 0x00,
    0x00, 0x00, 0x00, 0x7E, 0x18, 0x3C, 0x66, 0x66, 0x66, 0x3C, 0x18, 0x7E, 0x00, 0x00, 0x00, 0x00,
    0x00, 0x00, 0x00, 0x38, 0x6C, 0xC6, 0xC6, 0xFE, 0xC6, 0xC6, 0x6C, 0x38, 0x00, 0x00, 0x00, 0x00,
    0x00, 0x00, 0x38, 0x6C, 0xC6, 0xC6, 0xC6, 0x6C, 0x6C, 0x6C, 0x6C, 0xEE, 0x00, 0x00, 0x00, 0x00,
    0x00, 0x00, 0x1E, 0x30, 0x18, 0x0C, 0x3E, 0x66, 0x66, 0x66, 0x66, 0x3C, 0x00, 0x00, 0x00, 0x00,
    0x00, 0x00, 0x00, 0x00, 0x00, 0x7E, 0xDB, 0xDB, 0xDB, 0x7E, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
    0x00, 0x00, 0x00, 0x03, 0x06, 0x7E, 0xCF, 0xDB, 0xF3, 0x7E, 0x60, 0xC0, 0x00, 0x00, 0x00, 0x00,
    0x00, 0x00, 0x1C, 0x30, 0x60, 0x60, 0x7C, 0x60, 0x60, 0x60, 0x30, 0x1C, 0x00, 0x00, 0x00, 0x00,
    0x00, 0x00, 0x00, 0x7C, 0xC6, 0xC6, 0xC6, 0xC6, 0xC6, 0xC6, 0xC6, 0xC6, 0x00, 0x00, 0x00, 0x00,
    0x00, 0x00, 0x00, 0x00, 0xFE, 0x00, 0x00, 0xFE, 0x00, 0x00, 0xFE, 0x00, 0x00, 0x00, 0x00, 0x00,
    0x00, 0x00, 0x00, 0x00, 0x18, 0x18, 0x7E, 0x18, 0x18, 0x00, 0x00, 0xFF, 0x00, 0x00, 0x00, 0x00,
    0x00, 0x00, 0x00, 0x30, 0x18, 0x0C, 0x06, 0x0C, 0x18, 0x30, 0x00, 0x7E, 0x00, 0x00, 0x00, 0x00,
    0x00, 0x00, 0x00, 0x0C, 0x18, 0x30, 0x60, 0x30, 0x18, 0x0C, 0x00, 0x7E, 0x00, 0x00, 0x00, 0x00,
    0x00, 0x00, 0x0E, 0x1B, 0x1B, 0x18, 0x18, 0x18, 0x18, 0x18, 0x18, 0x18, 0x18, 0x18, 0x18, 0x18,
    0x18, 0x18, 0x18, 0x18, 0x18, 0x18, 0x18, 0x18, 0xD8, 0xD8, 0xD8, 0x70, 0x00, 0x00, 0x00, 0x00,
    0x00, 0x00, 0x00, 0x00, 0x18, 0x18, 0x00, 0x7E, 0x00, 0x18, 0x18, 0x00, 0x00, 0x00, 0x00, 0x00,
    0x00, 0x00, 0x00, 0x00, 0x00, 0x76, 0xDC, 0x00, 0x76, 0xDC, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
    0x00, 0x38, 0x6C, 0x6C, 0x38, 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, 0x00,
    0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x18, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
    0x00, 0x0F, 0x0C, 0x0C, 0x0C, 0x0C, 0x0C, 0xEC, 0x6C, 0x6C, 0x3C, 0x1C, 0x00, 0x00, 0x00, 0x00,
    0x00, 0xD8, 0x6C, 0x6C, 0x6C, 0x6C, 0x6C, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
    0x00, 0x70, 0x98, 0x30, 0x60, 0xC8, 0xF8, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
    0x00, 0x00, 0x00, 0x00, 0x7C, 0x7C, 0x7C, 0x7C, 0x7C, 0x7C, 0x7C, 0x00, 0x00, 0x00, 0x00, 0x00,
    0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00
};

/*************************************************************************
MAIN/DEMO ROUTINES
*************************************************************************/
static void dump(unsigned char *regs, unsigned count)
{
        unsigned i;

        i = 0;
        printf("\t");
        for(; count != 0; count--)
        {
                printf("0x%02X,", *regs);
                i++;
                if(i >= 8)
                {
                        i = 0;
                        printf("\n\t");
                }
                else
                        printf(" ");
                regs++;
        }
        printf("\n");
}
/*************************************************************************
*************************************************************************/
void dump_regs(unsigned char *regs)
{
        printf("unsigned char g_mode[] =\n");
        printf("{\n");
/* dump MISCELLANEOUS reg */
        printf("/* MISC */\n");
        printf("\t0x%02X,\n", *regs);
        regs++;
/* dump SEQUENCER regs */
        printf("/* SEQ */\n");
        dump(regs, VGA_NUM_SEQ_REGS);
        regs += VGA_NUM_SEQ_REGS;
/* dump CRTC regs */
        printf("/* CRTC */\n");
        dump(regs, VGA_NUM_CRTC_REGS);
        regs += VGA_NUM_CRTC_REGS;
```

```
/* dump GRAPHICS CONTROLLER regs */
        printf("/* GC */\n");
        dump(regs, VGA_NUM_GC_REGS);
        regs += VGA_NUM_GC_REGS;
/* dump ATTRIBUTE CONTROLLER regs */
        printf("/* AC */\n");
        dump(regs, VGA_NUM_AC_REGS);
        regs += VGA_NUM_AC_REGS;
        printf("};\n");
}
/*************************************************************************
*************************************************************************/
void read_regs(unsigned char *regs)
{
        unsigned i;

/* read MISCELLANEOUS reg */
        *regs = inportb(VGA_MISC_READ);
        regs++;
/* read SEQUENCER regs */
        for(i = 0; i < VGA_NUM_SEQ_REGS; i++)
        {
                outportb(VGA_SEQ_INDEX, i);
                *regs = inportb(VGA_SEQ_DATA);
                regs++;
        }
/* read CRTC regs */
        for(i = 0; i < VGA_NUM_CRTC_REGS; i++)
        {
                outportb(VGA_CRTC_INDEX, i);
                *regs = inportb(VGA_CRTC_DATA);
                regs++;
        }
/* read GRAPHICS CONTROLLER regs */
        for(i = 0; i < VGA_NUM_GC_REGS; i++)
        {
                outportb(VGA_GC_INDEX, i);
                *regs = inportb(VGA_GC_DATA);
                regs++;
        }
/* read ATTRIBUTE CONTROLLER regs */
        for(i = 0; i < VGA_NUM_AC_REGS; i++)
        {
                (void)inportb(VGA_INSTAT_READ);
                outportb(VGA_AC_INDEX, i);
                *regs = inportb(VGA_AC_READ);
                regs++;
        }
/* lock 16-color palette and unblank display */
        (void)inportb(VGA_INSTAT_READ);
        outportb(VGA_AC_INDEX, 0x20);
}
/*************************************************************************
*************************************************************************/
void write_regs(unsigned char *regs)
{
        unsigned i;

/* write MISCELLANEOUS reg */
        outportb(VGA_MISC_WRITE, *regs);
        regs++;
/* write SEQUENCER regs */
        for(i = 0; i < VGA_NUM_SEQ_REGS; i++)
        {
                outportb(VGA_SEQ_INDEX, i);
                outportb(VGA_SEQ_DATA, *regs);
                regs++;
        }
```

```
/* unlock CRTC registers */
        outportb(VGA_CRTC_INDEX, 0x03);
        outportb(VGA_CRTC_DATA, inportb(VGA_CRTC_DATA) | 0x80);
        outportb(VGA_CRTC_INDEX, 0x11);
        outportb(VGA_CRTC_DATA, inportb(VGA_CRTC_DATA) & ~0x80);
/* make sure they remain unlocked */
        regs[0x03] |= 0x80;
        regs[0x11] &= ~0x80;
/* write CRTC regs */
        for(i = 0; i < VGA_NUM_CRTC_REGS; i++)
        {
                outportb(VGA_CRTC_INDEX, i);
                outportb(VGA_CRTC_DATA, *regs);
                regs++;
        }
/* write GRAPHICS CONTROLLER regs */
        for(i = 0; i < VGA_NUM_GC_REGS; i++)
        {
                outportb(VGA_GC_INDEX, i);
                outportb(VGA_GC_DATA, *regs);
                regs++;
        }
/* write ATTRIBUTE CONTROLLER regs */
        for(i = 0; i < VGA_NUM_AC_REGS; i++)
        {
                (void)inportb(VGA_INSTAT_READ);
                outportb(VGA_AC_INDEX, i);
                outportb(VGA_AC_WRITE, *regs);
                regs++;
        }
/* lock 16-color palette and unblank display */
        (void)inportb(VGA_INSTAT_READ);
        outportb(VGA_AC_INDEX, 0x20);
}
/****************************************************************************
****************************************************************************/
static void set_plane(unsigned p)
{
        unsigned char pmask;

        p &= 3;
        pmask = 1 << p;
/* set read plane */
        outportb(VGA_GC_INDEX, 4);
        outportb(VGA_GC_DATA, p);
/* set write plane */
        outportb(VGA_SEQ_INDEX, 2);
        outportb(VGA_SEQ_DATA, pmask);
}
```

```
/*****************************************************************************
VGA framebuffer is at A000:0000, B000:0000, or B800:0000
depending on bits in GC 6
*****************************************************************************/
static unsigned get_fb_seg(void)
{
        unsigned seg;

        outportb(VGA_GC_INDEX, 6);
        seg = inportb(VGA_GC_DATA);
        seg >>= 2;
        seg &= 3;
        switch(seg)
        {
        case 0:
        case 1:
                seg = 0xA000;
                break;
        case 2:
                seg = 0xB000;
                break;
        case 3:
                seg = 0xB800;
                break;
        }
        return seg;
}
/*****************************************************************************
*****************************************************************************/
static void vmemwr(unsigned dst_off, unsigned char *src, unsigned count)
{
        _vmemwr(get_fb_seg(), dst_off, src, count);
}
/*****************************************************************************
*****************************************************************************/
static void vpokeb(unsigned off, unsigned val)
{
        pokeb(get_fb_seg(), off, val);
}
/*****************************************************************************
*****************************************************************************/
static unsigned vpeekb(unsigned off)
{
        return peekb(get_fb_seg(), off);
}
/*****************************************************************************
write font to plane P4 (assuming planes are named P1, P2, P4, P8)
*****************************************************************************/
static void write_font(unsigned char *buf, unsigned font_height)
{
        unsigned char seq2, seq4, gc4, gc5, gc6;
        unsigned i;

/* save registers
set_plane() modifies GC 4 and SEQ 2, so save them as well */
        outportb(VGA_SEQ_INDEX, 2);
        seq2 = inportb(VGA_SEQ_DATA);

        outportb(VGA_SEQ_INDEX, 4);
        seq4 = inportb(VGA_SEQ_DATA);
/* turn off even-odd addressing (set flat addressing)
assume: chain-4 addressing already off */
        outportb(VGA_SEQ_DATA, seq4 | 0x04);

        outportb(VGA_GC_INDEX, 4);
        gc4 = inportb(VGA_GC_DATA);
```

```
        outportb(VGA_GC_INDEX, 5);
        gc5 = inportb(VGA_GC_DATA);
/* turn off even-odd addressing */
        outportb(VGA_GC_DATA, gc5 & ~0x10);

        outportb(VGA_GC_INDEX, 6);
        gc6 = inportb(VGA_GC_DATA);
/* turn off even-odd addressing */
        outportb(VGA_GC_DATA, gc6 & ~0x02);
/* write font to plane P4 */
        set_plane(2);
/* write font 0 */
        for(i = 0; i < 256; i++)
        {
                vmemwr(16384u * 0 + i * 32, buf, font_height);
                buf += font_height;
        }
#if 0
/* write font 1 */
        for(i = 0; i < 256; i++)
        {
                vmemwr(16384u * 1 + i * 32, buf, font_height);
                buf += font_height;
        }
#endif
/* restore registers */
        outportb(VGA_SEQ_INDEX, 2);
        outportb(VGA_SEQ_DATA, seq2);
        outportb(VGA_SEQ_INDEX, 4);
        outportb(VGA_SEQ_DATA, seq4);
        outportb(VGA_GC_INDEX, 4);
        outportb(VGA_GC_DATA, gc4);
        outportb(VGA_GC_INDEX, 5);
        outportb(VGA_GC_DATA, gc5);
        outportb(VGA_GC_INDEX, 6);
        outportb(VGA_GC_DATA, gc6);
}
/****************************************************************************
****************************************************************************/
static void (*g_write_pixel)(unsigned x, unsigned y, unsigned c);
static unsigned g_wd, g_ht;

static void write_pixel1(unsigned x, unsigned y, unsigned c)
{
        unsigned wd_in_bytes;
        unsigned off, mask;

        c = (c & 1) * 0xFF;
        wd_in_bytes = g_wd / 8;
        off = wd_in_bytes * y + x / 8;
        x = (x & 7) * 1;
        mask = 0x80 >> x;
        vpokeb(off, (vpeekb(off) & ~mask) | (c & mask));
}
/****************************************************************************
****************************************************************************/
static void write_pixel2(unsigned x, unsigned y, unsigned c)
{
        unsigned wd_in_bytes, off, mask;

        c = (c & 3) * 0x55;
        wd_in_bytes = g_wd / 4;
        off = wd_in_bytes * y + x / 4;
        x = (x & 3) * 2;
        mask = 0xC0 >> x;
        vpokeb(off, (vpeekb(off) & ~mask) | (c & mask));
}
```

```
/*****************************************************************************
*****************************************************************************/
static void write_pixel4p(unsigned x, unsigned y, unsigned c)
{
        unsigned wd_in_bytes, off, mask, p, pmask;

        wd_in_bytes = g_wd / 8;
        off = wd_in_bytes * y + x / 8;
        x = (x & 7) * 1;
        mask = 0x80 >> x;
        pmask = 1;
        for(p = 0; p < 4; p++)
        {
                set_plane(p);
                if(pmask & c)
                        vpokeb(off, vpeekb(off) | mask);
                else
                        vpokeb(off, vpeekb(off) & ~mask);
                pmask <<= 1;
        }
}
/*****************************************************************************
*****************************************************************************/
static void write_pixel8(unsigned x, unsigned y, unsigned c)
{
        unsigned wd_in_bytes;
        unsigned off;

        wd_in_bytes = g_wd;
        off = wd_in_bytes * y + x;
        vpokeb(off, c);
}
/*****************************************************************************
*****************************************************************************/
static void write_pixel8x(unsigned x, unsigned y, unsigned c)
{
        unsigned wd_in_bytes;
        unsigned off;

        wd_in_bytes = g_wd / 4;
        off = wd_in_bytes * y + x / 4;
        set_plane(x & 3);
        vpokeb(off, c);
}
/*****************************************************************************
*****************************************************************************/
static void draw_x(void)
{
        unsigned x, y;

/* clear screen */
        for(y = 0; y < g_ht; y++)
                for(x = 0; x < g_wd; x++)
                        g_write_pixel(x, y, 0);
/* draw 2-color X */
        for(y = 0; y < g_ht; y++)
        {
                g_write_pixel((g_wd - g_ht) / 2 + y, y, 1);
                g_write_pixel((g_ht + g_wd) / 2 - y, y, 2);
        }
        getch();
}
```

```
/****************************************************************************
READ AND DUMP VGA REGISTER VALUES FOR CURRENT VIDEO MODE
This is where g_40x25_text[], g_80x50_text[], etc. came from :)
****************************************************************************/
void dump_state(void)
{
        unsigned char state[VGA_NUM_REGS];

        read_regs(state);
        dump_regs(state);
}
/****************************************************************************
SET TEXT MODES
****************************************************************************/
void set_text_mode(int hi_res)
{
        unsigned rows, cols, ht, i;

        if(hi_res)
        {
                write_regs(g_90x60_text);
                cols = 90;
                rows = 60;
                ht = 8;
        }
        else
        {
                write_regs(g_80x25_text);
                cols = 80;
                rows = 25;
                ht = 16;
        }
/* set font */
        if(ht >= 16)
                write_font(g_8x16_font, 16);
        else
                write_font(g_8x8_font, 8);
/* tell the BIOS what we've done, so BIOS text output works OK */
        pokew(0x40, 0x4A, cols);  /* columns on screen */
        pokew(0x40, 0x4C, cols * rows * 2); /* framebuffer size */
        pokew(0x40, 0x50, 0);             /* cursor pos'n */
        pokeb(0x40, 0x60, ht - 1);        /* cursor shape */
        pokeb(0x40, 0x61, ht - 2);
        pokeb(0x40, 0x84, rows - 1);      /* rows on screen - 1 */
        pokeb(0x40, 0x85, ht);            /* char height */
/* set white-on-black attributes for all text */
        for(i = 0; i < cols * rows; i++)
                pokeb(0xB800, i * 2 + 1, 7);
}
/****************************************************************************
DEMO GRAPHICS MODES
****************************************************************************/
static void demo_graphics(void)
{
        printf("Screen-clear in 16-color mode will be VERY SLOW\n"
                "Press a key to continue\n");
        getch();
/* 4-color */
        write_regs(g_320x200x4);
        g_wd = 320;
        g_ht = 200;
        g_write_pixel = write_pixel2;
        draw_x();
/* 16-color */
        write_regs(g_640x480x16);
        g_wd = 640;
        g_ht = 480;
        g_write_pixel = write_pixel4p;
```

```
        draw_x();
/* 256-color */
        write_regs(g_320x200x256);
        g_wd = 320;
        g_ht = 200;
        g_write_pixel = write_pixel8;
        draw_x();
/* 256-color Mode-X */
        write_regs(g_320x200x256_modex);
        g_wd = 320;
        g_ht = 200;
        g_write_pixel = write_pixel8x;
        draw_x();
/* go back to 80x25 text mode */
        set_text_mode(0);
}
/*************************************************************************
*************************************************************************/
static unsigned char reverse_bits(unsigned char arg)
{
        unsigned char ret_val = 0;

        if(arg & 0x01)
                ret_val |= 0x80;
        if(arg & 0x02)
                ret_val |= 0x40;
        if(arg & 0x04)
                ret_val |= 0x20;
        if(arg & 0x08)
                ret_val |= 0x10;
        if(arg & 0x10)
                ret_val |= 0x08;
        if(arg & 0x20)
                ret_val |= 0x04;
        if(arg & 0x40)
                ret_val |= 0x02;
        if(arg & 0x80)
                ret_val |= 0x01;
        return ret_val;
}
/*************************************************************************
512-CHARACTER FONT
*************************************************************************/
static void font512(void)
{
/* Turbo C++ 1.0 seems to 'lose' any data declared 'static const' */
        /*static*/ const char msg1[] = "!txet sdrawkcaB";
        /*static*/ const char msg2[] = "?rorrim a toG";
/**/
        unsigned char seq2, seq4, gc4, gc5, gc6;
        unsigned font_height, i, j;

/* start in 80x25 text mode */
        set_text_mode(0);
/* code pasted in from write_font():
save registers
set_plane() modifies GC 4 and SEQ 2, so save them as well */
        outportb(VGA_SEQ_INDEX, 2);
        seq2 = inportb(VGA_SEQ_DATA);

        outportb(VGA_SEQ_INDEX, 4);
        seq4 = inportb(VGA_SEQ_DATA);
/* turn off even-odd addressing (set flat addressing)
assume: chain-4 addressing already off */
        outportb(VGA_SEQ_DATA, seq4 | 0x04);

        outportb(VGA_GC_INDEX, 4);
        gc4 = inportb(VGA_GC_DATA);
```

```
        outportb(VGA_GC_INDEX, 5);
        gc5 = inportb(VGA_GC_DATA);
/* turn off even-odd addressing */
        outportb(VGA_GC_DATA, gc5 & ~0x10);

        outportb(VGA_GC_INDEX, 6);
        gc6 = inportb(VGA_GC_DATA);
/* turn off even-odd addressing */
        outportb(VGA_GC_DATA, gc6 & ~0x02);
/* write font to plane P4 */
        set_plane(2);
/* this is different from write_font():
use font 1 instead of font 0, and use it for BACKWARD text */
        font_height = 16;
        for(i = 0; i < 256; i++)
        {
                for(j = 0; j < font_height; j++)
                {
                        vpokeb(16384u * 1 + 32 * i + j,
                                reverse_bits(
                                        g_8x16_font[font_height * i + j]));
                }
        }
/* restore registers */
        outportb(VGA_SEQ_INDEX, 2);
        outportb(VGA_SEQ_DATA, seq2);
        outportb(VGA_SEQ_INDEX, 4);
        outportb(VGA_SEQ_DATA, seq4);
        outportb(VGA_GC_INDEX, 4);
        outportb(VGA_GC_DATA, gc4);
        outportb(VGA_GC_INDEX, 5);
        outportb(VGA_GC_DATA, gc5);
        outportb(VGA_GC_INDEX, 6);
        outportb(VGA_GC_DATA, gc6);
/* now: sacrifice attribute bit b3 (foreground intense color)
use it to select characters 256-511 in the second font */
        outportb(VGA_SEQ_INDEX, 3);
        outportb(VGA_SEQ_DATA, 4);
/* xxx - maybe re-program 16-color palette here
so attribute bit b3 is no longer used for 'intense' */
        for(i = 0; i < sizeof(msg1); i++)
        {
                vpokeb((80 * 8  + 40 + i) * 2 + 0, msg1[i]);
/* set attribute bit b3 for backward font */
                vpokeb((80 * 8  + 40 + i) * 2 + 1, 0x0F);
        }
        for(i = 0; i < sizeof(msg2); i++)
        {
                vpokeb((80 * 16 + 40 + i) * 2 + 0, msg2[i]);
                vpokeb((80 * 16 + 40 + i) * 2 + 1, 0x0F);
        }
}
/*************************************************************************
*************************************************************************/
int main(int arg_c, char *arg_v[])
{
        //dump_state();
        //set_text_mode(arg_c > 1);
        //demo_graphics();
        font512();
        return 0;
}
```