Day 4

首先吐个槽，haribote.sys不能正常生成，原因是make无法执行指令 `copy /B asmhead.bin+bootpack.hrb haribote.sys`

```
D:\git-repos\30dayMakeOS\OS\tolset\harib01a>make

D:\git-repos\30dayMakeOS\OS\tolset\harib01a>..\z_tools\make.exe
../z_tools/make.exe -r img
make.exe[1]: Entering directory `D:/git-repos/30dayMakeOS/OS/tolset/harib01a'
../z_tools/make.exe -r haribote.img
make.exe[2]: Entering directory `D:/git-repos/30dayMakeOS/OS/tolset/harib01a'
../z_tools/cc1.exe -I../z_tools/haribote/ -Os -Wall -quiet -o bootpack.gas bootpack.c
../z_tools/gas2nask.exe -a bootpack.gas bootpack.nas
../z_tools/nask.exe bootpack.nas bootpack.obj bootpack.lst
../z_tools/obj2bim.exe @../z_tools/haribote/haribote.rul out:bootpack.bim stack:3136k map:bootpack.map \
        bootpack.obj naskfunc.obj
../z_tools/bim2hrb.exe bootpack.bim bootpack.hrb 0
copy /B asmhead.bin+bootpack.hrb haribote.sys
process_begin: CreateProcess((null), copy /B asmhead.bin+bootpack.hrb haribote.sys, ...) failed.
make (e=2): 系统找不到指定的文件。
make.exe[2]: *** [haribote.sys] Error 2
make.exe[2]: Leaving directory `D:/git-repos/30dayMakeOS/OS/tolset/harib01a'
make.exe[1]: *** [img] Error 2
make.exe[1]: Leaving directory `D:/git-repos/30dayMakeOS/OS/tolset/harib01a'
..\z_tools\make.exe: *** [default] Error 2
```

但是这个指令直接在cmd里跑是可以正常执行的。目前暂时的解决方案是遇到错误后手动执行一下

`copy /B asmhead.bin+bootpack.hrb haribote.sys`

然后再重新 `make` 就ok了

Phase 1

SubPhase 1

用函数的方式实现C语言对内存的写入

一如既往的，我们再naskfunc.nas当中添加一些代码。这次我们将会学会如何在汇编语言下获取函数调用的参数

```
_write_mem8:    ; void write_mem8(int addr, int data);
                MOV             ECX,[ESP+4]             ; ESP+4是第一个参数的地址
                MOV             AL,[ESP+8]              ; ESP+8是第二个参数的地址
                                        ; int是4字节
                MOV             [ECX],AL
                RET
```

调用的时候只要

```
write_mem8(地址, 数据);
```

就可以修改内存了

SubPhase 2

尝试输出一些东西

通过阅读得知VRAM的内容会反映到屏幕上。因此我们可以通过修改VRAM来显示图像内容。总共16中颜色，编号从0到15。我们直接一步到位！来一段会移动的不同色的条纹图案！

以下是我乱搞出来的代码

```
void io_hlt(void);
void write_mem8(int addr, int data);
void wait(int x) {
    x *= 1000;
    while (x--);
}
void HariMain(void)
{
        int i;
    int cnt;
    while (1) {
        for (i = 0xa0000; i <= 0xaffff; i++) {
                write_mem8(i, ((i - cnt) >> 1) & 0x0f);
        }
        cnt = (cnt + 1) & 0xffff;
        wait(100000);
    }
}
```

编译，扔到virtualbox中运行

效果见此链接http://v.douyin.com/2JvjFb （配了个背景音乐qvq）

指针对于我们来说并不是什么难事，因此书上对于指针的部分大可以跳过去。

使用指针进行改写，如下

```
void io_hlt(void);
void write_mem8(int addr, int data);
void wait(int x) {
    x *= 1000;
    while (x--);
}
void HariMain(void)
{
        char *i;
    int cnt;
    while (1) {
        for (i = (char*)0xa0000; i <= (char*)0xaffff; i++) {
                *i = (((int)i - cnt) >> 1) & 0x0f;  // 对i进行类型转换，否则不能过编
        }
        cnt = (cnt + 1) & 0xffff;
        wait(100000);
    }
}
```

Phase 2

色号设定

VRAM当中只是写了每个像素对应的颜色编号。想要具体的指定每个颜色编号对应什么颜色，我们需要设置色板。分析一波代码

```
void set_palette(int start, int end, unsigned char *rgb)
{
```

```
        int i, eflags;
        eflags = io_load_eflags();      /* 割り込み許可フラグの値を記録する */
        io_cli();                                       /* 許可フラグを0にして割り込み禁止にする
*/
        io_out8(0x03c8, start);
        for (i = start; i <= end; i++) {
                io_out8(0x03c9, rgb[0] / 4);
                io_out8(0x03c9, rgb[1] / 4);
                io_out8(0x03c9, rgb[2] / 4);
                rgb += 3;
        }
        io_store_eflags(eflags);        /* 割り込み許可フラグを元に戻す */
        return;
}
```

程序先将中断许可标志保存下来（有点类似于中断的保存现场？），然后禁止中断，并向显示数模转换器发送48个电信号，设定每个色号的具体颜色。也即是说颜色其实是在数模转换器当中设置的。那为什么RGB的每个分量都要除以4呢？这是因为原来的最大值是255，也就是范围大小是256，显卡中的DA转换器接受的是6位分量，所以要除以4

完成了色号的设置之后我们把终端许可标志复原（恢复现场）

分析一下 `io_load_eflags` 。这个函数是我们第一个具有返回值的函数。根据c语言的规约，eax中的值将作为ret时函数的返回值。因此我们把返回值mov到eax当中就行了

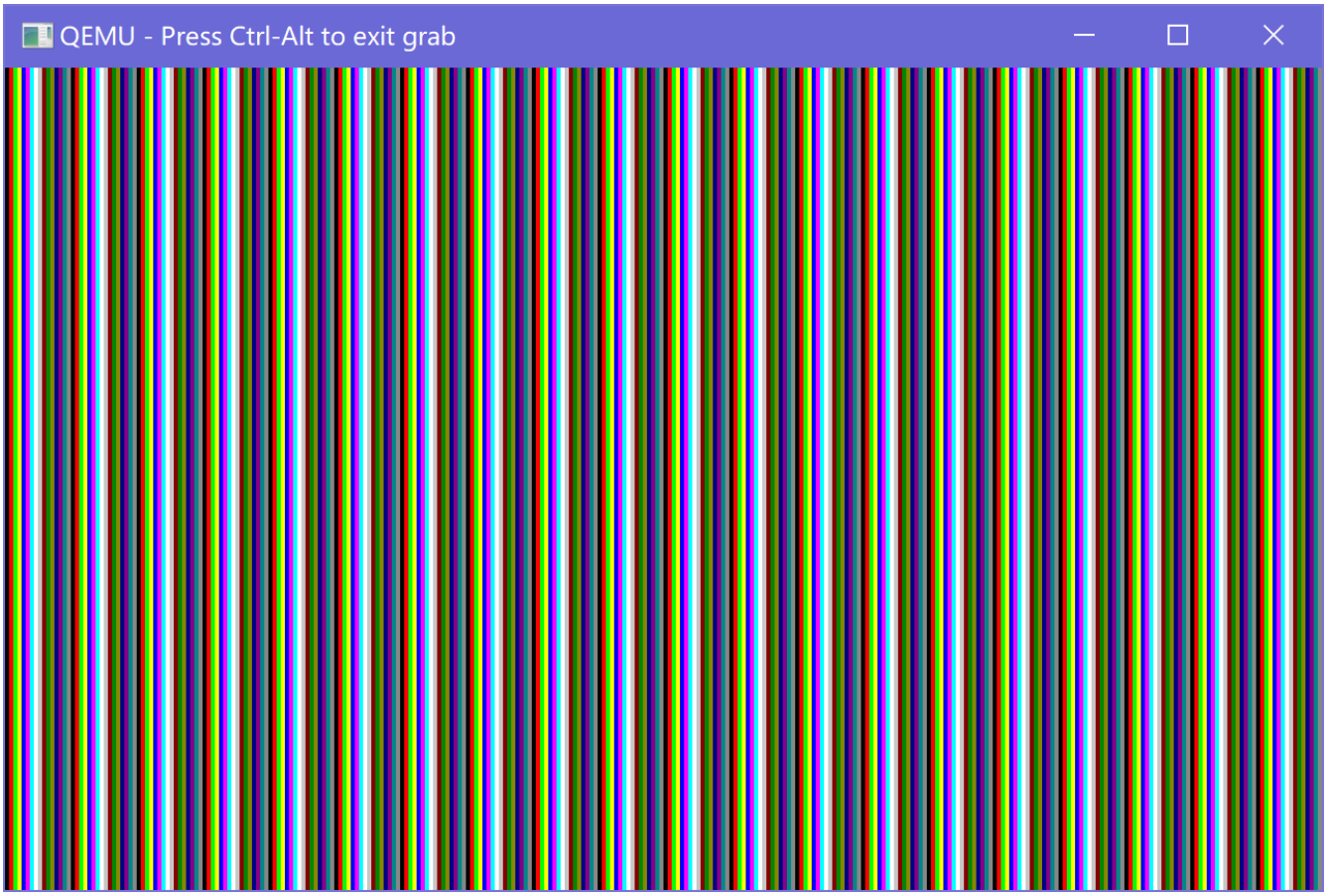
`io_out` 函数是通过总线向其他设备直接送数据，主要是通过汇编指令 `out addr, data` 来实现的。

然后我们再看看 `init_palette` 函数

函数首先定义了一个静态char数组。静态char数组将被翻译成汇编的DB语句，里面是每种颜色的各个分量。然后调用 `set_palette` 将颜色设置发送到显卡。

我们试试修改色板之后的程序。

如下



Phase 3

绘制矩形。

当前画面模式（320x200）下像素点(x,y)的对应VRAM地址是`0xa0000 + x + y * 320`，我们只需要一个嵌套循环就可以绘制矩形啦！我手撸一个这样的函数。

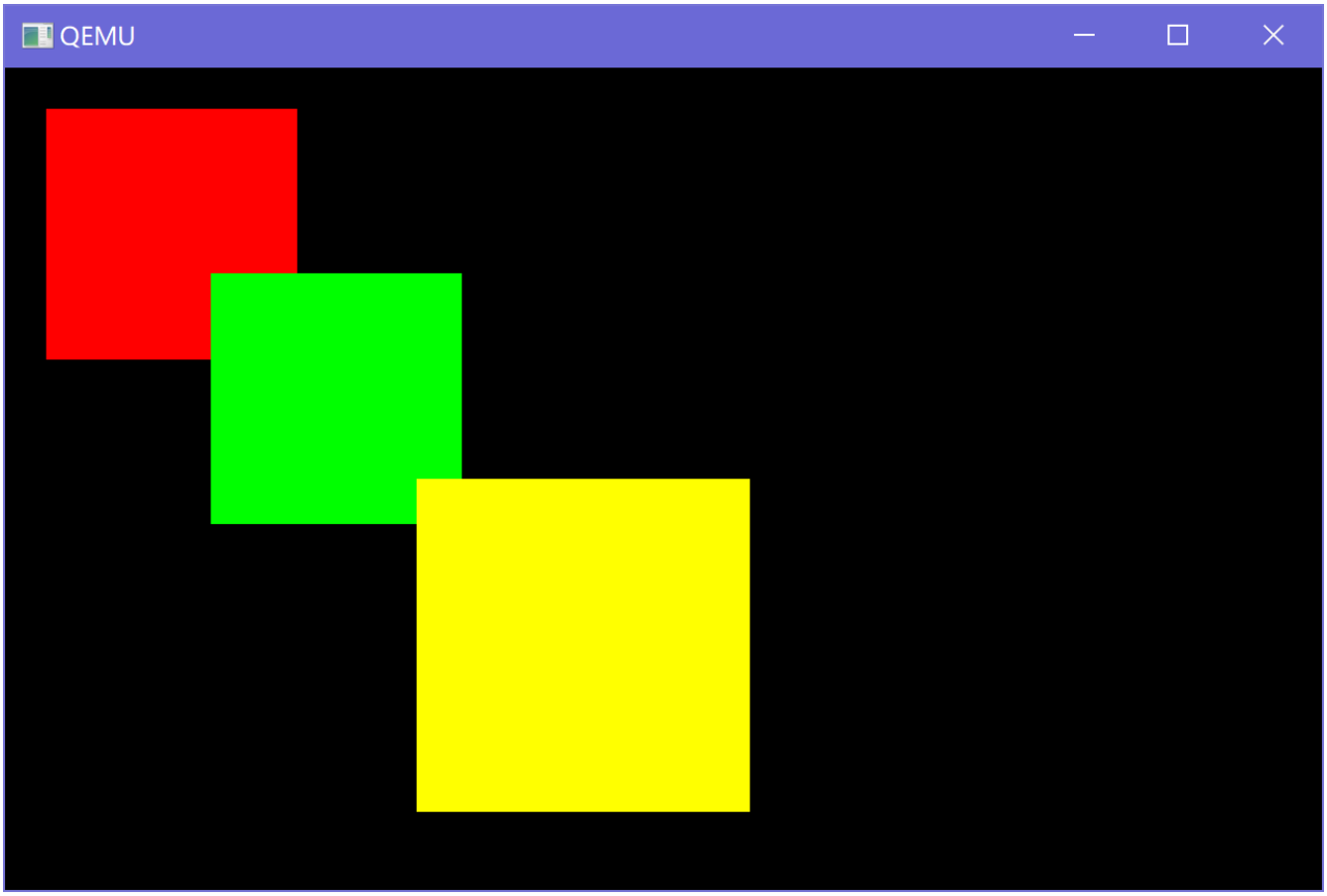
```
void boxfill(unsigned char *vram, int width, unsigned char color, int x0, int y0, int x1, int
y1) {
    int x, y;
    for (y = y0; y <= y1; y++) {
        for (int x = x0; x <= x1; x++) {
            vram[x + y * width] = c;
        }
    }
    return;
}
```

测试程序：

```
void io_hlt(void);
void io_cli(void);
void io_out8(int port, int data);
int io_load_eflags(void);
void io_store_eflags(int eflags);

/* 実は同じソースファイルに書いてあっても、定義する前に使うのなら、
        やっぱり宣言しておかないといけない。 */

void init_palette(void);
void set_palette(int start, int end, unsigned char *rgb);
void boxfill(unsigned char *vram, int width, unsigned char color, int x0, int y0, int x1, int
y1);
void HariMain(void)
{
        int i; /* 変数宣言。iという変数は、32ビットの整数型 */

        init_palette(); /* パレットを設定 */
        char *p = (char*)0xa0000;
        boxfill(p, 320, 1, 10, 10, 70, 70);
        boxfill(p, 320, 2, 50, 50, 110, 110);
        boxfill(p, 320, 3, 100, 100, 180, 180);
        for (;;) {
                io_hlt();
        }
}
void boxfill(unsigned char *vram, int width, unsigned char color, int x0, int y0, int x1, int
y1) {
    int x, y;
    for (y = y0; y <= y1; y++) {
        for (x = x0; x <= x1; x++) {
            vram[x + y * width] = color;
        }
    }
    return;
}
void init_palette(void)
{
        static unsigned char table_rgb[16 * 3] = {
                0x00, 0x00, 0x00,       /*  0:黒 */
                0xff, 0x00, 0x00,       /*  1:明るい赤 */
                0x00, 0xff, 0x00,       /*  2:明るい緑 */
                0xff, 0xff, 0x00,       /*  3:明るい黄色 */
                0x00, 0x00, 0xff,       /*  4:明るい青 */
                0xff, 0x00, 0xff,       /*  5:明るい紫 */
```

```
                0x00, 0xff, 0xff,       /*  6:明るい水色 */
                0xff, 0xff, 0xff,       /*  7:白 */
                0xc6, 0xc6, 0xc6,       /*  8:明るい灰色 */
                0x84, 0x00, 0x00,       /*  9:暗い赤 */
                0x00, 0x84, 0x00,       /* 10:暗い緑 */
                0x84, 0x84, 0x00,       /* 11:暗い黄色 */
                0x00, 0x00, 0x84,       /* 12:暗い青 */
                0x84, 0x00, 0x84,       /* 13:暗い紫 */
                0x00, 0x84, 0x84,       /* 14:暗い水色 */
                0x84, 0x84, 0x84        /* 15:暗い灰色 */
        };
        set_palette(0, 15, table_rgb);
        return;

        /* static char 命令は、データにしか使えないけどDB命令相当 */
}

void set_palette(int start, int end, unsigned char *rgb)
{
        int i, eflags;
        eflags = io_load_eflags();      /* 割り込み許可フラグの値を記録する */
        io_cli();                                       /* 許可フラグを0にして割り込み禁止にする
*/
        io_out8(0x03c8, start);
        for (i = start; i <= end; i++) {
                io_out8(0x03c9, rgb[0] / 4);
                io_out8(0x03c9, rgb[1] / 4);
                io_out8(0x03c9, rgb[2] / 4);
                rgb += 3;
        }
        io_store_eflags(eflags);        /* 割り込み許可フラグを元に戻す */
        return;
}
```

测试结果



最后，我们尝试玩一玩boxfill，在书上的基础上绘制一个现代微软徽标吧！

代码如下

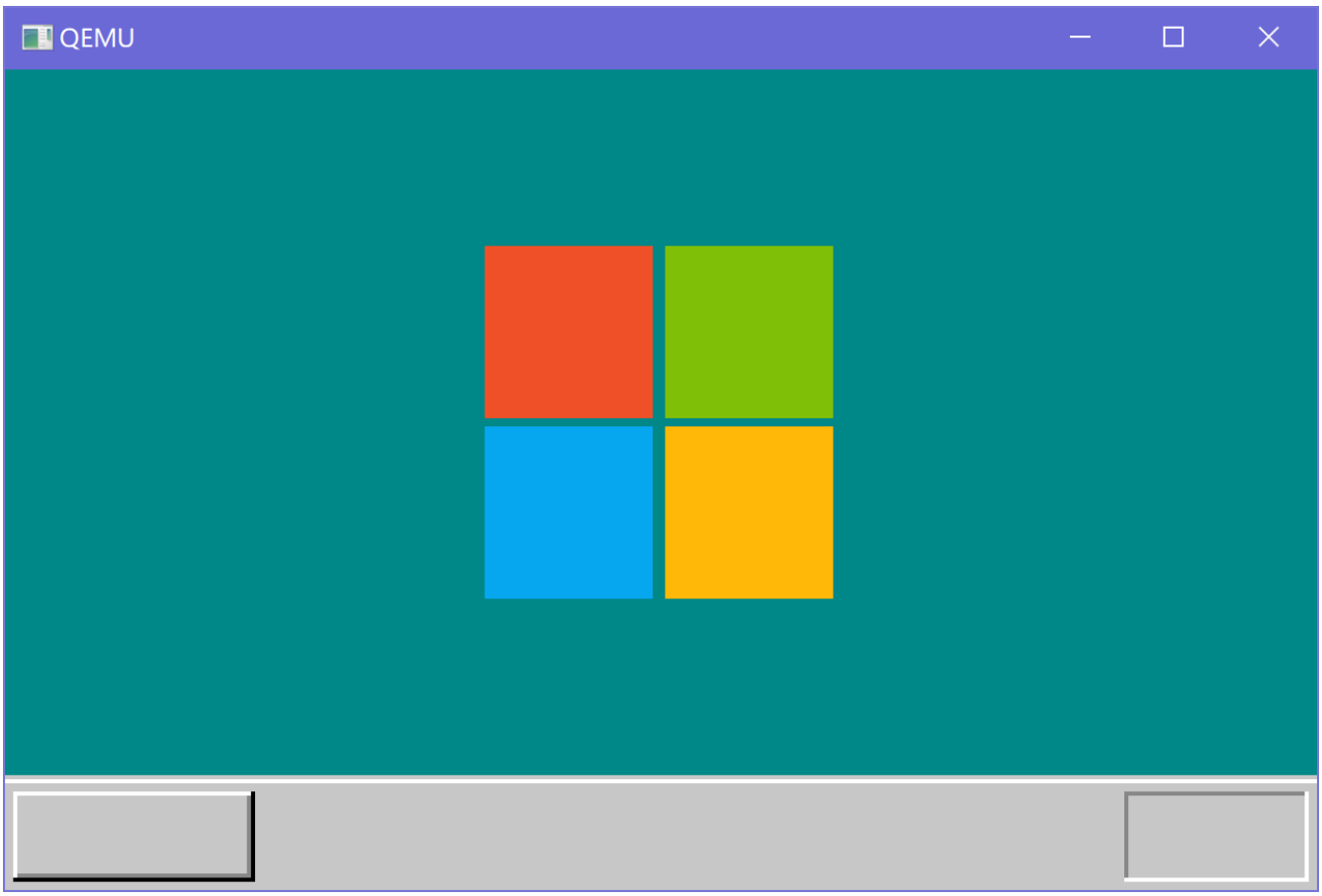
```
void io_hlt(void);
void io_cli(void);
void io_out8(int port, int data);
int io_load_eflags(void);
void io_store_eflags(int eflags);

void init_palette(void);
void set_palette(int start, int end, unsigned char *rgb);
void boxfill8(unsigned char *vram, int xsize, unsigned char c, int x0, int y0, int x1, int
y1);

#define COL8_000000             0
#define COL8_F35325             1
#define COL8_81BC06             2
#define COL8_FFBA08             3
#define COL8_05A6F0             4
#define COL8_FF00FF             5
#define COL8_00FFFF             6
#define COL8_FFFFFF             7
```

```
#define COL8_C6C6C6             8
#define COL8_840000             9
#define COL8_008400             10
#define COL8_848400             11
#define COL8_000084             12
#define COL8_840084             13
#define COL8_008484             14
#define COL8_848484             15

void HariMain(void)
{
        char *vram;
        int xsize, ysize;

        init_palette();
        vram = (char *) 0xa0000;
        xsize = 320;
        ysize = 200;

        boxfill8(vram, xsize, COL8_008484,  0,         0,          xsize -  1, ysize - 29);
        boxfill8(vram, xsize, COL8_C6C6C6,  0,         ysize - 28, xsize -  1, ysize - 28);
        boxfill8(vram, xsize, COL8_FFFFFF,  0,         ysize - 27, xsize -  1, ysize - 27);
        boxfill8(vram, xsize, COL8_C6C6C6,  0,         ysize - 26, xsize -  1, ysize -  1);

        boxfill8(vram, xsize, COL8_FFFFFF,  3,         ysize - 24, 59,         ysize - 24);
        boxfill8(vram, xsize, COL8_FFFFFF,  2,         ysize - 24,  2,         ysize -  4);
        boxfill8(vram, xsize, COL8_848484,  3,         ysize -  4, 59,         ysize -  4);
        boxfill8(vram, xsize, COL8_848484, 59,         ysize - 23, 59,         ysize -  5);
        boxfill8(vram, xsize, COL8_000000,  2,         ysize -  3, 59,         ysize -  3);
        boxfill8(vram, xsize, COL8_000000, 60,         ysize - 24, 60,         ysize -  3);

        boxfill8(vram, xsize, COL8_848484, xsize - 47, ysize - 24, xsize -  4, ysize - 24);
        boxfill8(vram, xsize, COL8_848484, xsize - 47, ysize - 23, xsize - 47, ysize -  4);
        boxfill8(vram, xsize, COL8_FFFFFF, xsize - 47, ysize -  3, xsize -  4, ysize -  3);
        boxfill8(vram, xsize, COL8_FFFFFF, xsize -  3, ysize - 24, xsize -  3, ysize -  3);

    boxfill8(vram, xsize, COL8_F35325, xsize / 2 - 1 - 42, (ysize - 28) / 2 - 1 - 42, xsize /
2 - 1 - 2, (ysize - 28) / 2 - 2);
    boxfill8(vram, xsize, COL8_81BC06, xsize / 2 - 1 + 2, (ysize - 28) / 2 - 1 - 42, xsize /
2 - 1 + 42, (ysize - 28) / 2 - 2);
    boxfill8(vram, xsize, COL8_05A6F0, xsize / 2 - 1 - 42, (ysize - 28) / 2 - 1 + 2, xsize /
2 - 1 - 2, (ysize - 28) / 2 + 42);
    boxfill8(vram, xsize, COL8_FFBA08, xsize / 2 - 1 + 2, (ysize - 28) / 2 - 1 + 2, xsize / 2
- 1 + 42, (ysize - 28) / 2 + 42);

        for (;;) {
```

```
                io_hlt();
        }
}

void init_palette(void)
{
        static unsigned char table_rgb[16 * 3] = {
                0x00, 0x00, 0x00,       /*  0:黒 */
                0xf3, 0x53, 0x25,       /*  1:明るい赤 */
                0x81, 0xbc, 0x06,       /*  2:明るい緑 */
                0xff, 0xba, 0x08,       /*  3:明るい黄色 */
                0x05, 0xa6, 0xf0,       /*  4:明るい青 */
                0xff, 0x00, 0xff,       /*  5:明るい紫 */
                0x00, 0xff, 0xff,       /*  6:明るい水色 */
                0xff, 0xff, 0xff,       /*  7:白 */
                0xc6, 0xc6, 0xc6,       /*  8:明るい灰色 */
                0x84, 0x00, 0x00,       /*  9:暗い赤 */
                0x00, 0x84, 0x00,       /* 10:暗い緑 */
                0x84, 0x84, 0x00,       /* 11:暗い黄色 */
                0x00, 0x00, 0x84,       /* 12:暗い青 */
                0x84, 0x00, 0x84,       /* 13:暗い紫 */
                0x00, 0x84, 0x84,       /* 14:暗い水色 */
                0x84, 0x84, 0x84        /* 15:暗い灰色 */
        };
        set_palette(0, 15, table_rgb);
        return;

        /* static char 命令は、データにしか使えないけどDB命令相当 */
}

void set_palette(int start, int end, unsigned char *rgb)
{
        int i, eflags;
        eflags = io_load_eflags();      /* 割り込み許可フラグの値を記録する */
        io_cli();                                       /* 許可フラグを0にして割り込み禁止にする
*/
        io_out8(0x03c8, start);
        for (i = start; i <= end; i++) {
                io_out8(0x03c9, rgb[0] / 4);
                io_out8(0x03c9, rgb[1] / 4);
                io_out8(0x03c9, rgb[2] / 4);
                rgb += 3;
        }
        io_store_eflags(eflags);        /* 割り込み許可フラグを元に戻す */
        return;
}
```

```
void boxfill8(unsigned char *vram, int xsize, unsigned char c, int x0, int y0, int x1, int
y1)
{
        int x, y;
        for (y = y0; y <= y1; y++) {
                for (x = x0; x <= x1; x++)
                        vram[y * xsize + x] = c;
        }
        return;
}
```

效果如下：



PS：我改了色板，因为纯色实在不是很好看。

至此我们已经完成了今天的任务，知道了如何用c语言修改内存，学会了在屏幕上绘图并利用io命令向dac发送修改色板的指令。期待明天！