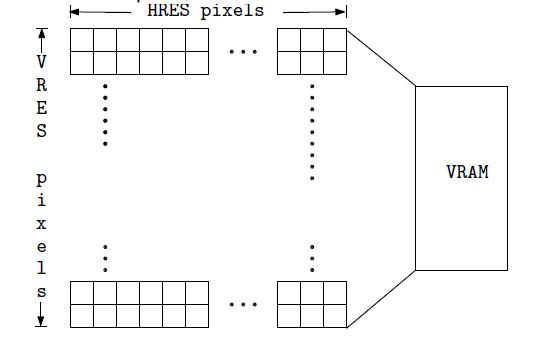
**Video Card**



* O ecrã do PC pode ser interpretado como uma matriz de pixéis, com HRES (Horizontal Resolution) pixeis na horizontal, e VRES (Vertical Resolution) pixeis na vertical
* A VRAM (Video RAM) guarda a informação que é “renderizada”(rendered) no ecrã, incluindo a cor de cada pixel

- A cor de cada pixel é obtida somando as três cores primárias (Vermelho, Verde, e Azul)

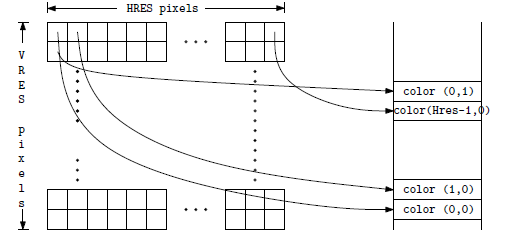
- Existem dois modos (pelo menos neste lab) de codificar a cor:

* **Modo Direto:** Guarda a cor de cada pixel na VRAM
* **Modo Indexado:** Em vez de a cor por pixel, guarda um índex numa tabela (um color map)

**Memory Models**

- O modelo de memória determina como a VRAM é organizada, ou seja, onde o valor de cada pixel é guardado na VRAM

- O modo que se usa no lab5 é o modo linear, em que cada pixel é guardado, da esquerda para a direita e de cima para baixo, por ordem na VRAM. Ou seja, é guardado primeiro o pixel no canto superior esquerdo, e depois o ao seu lado, assim continuando até chegar ao pixel do canto inferior direito



Para aceder ao pixel, precisamos de saber:

* o endereço base do frame buffer (local onde a VRAM está guardada)
* as coordenadas do pixel
* o número de bytes usado para codificar a cor

**3. The VBE Standard**

- A VBE (VESA BIOS Extensions) surge como necessidade para definir normas fixas para a placa gráfica. Define um conjunto de funções relacionas com esta e especifica como podem ser invocadas.

**3.1 Accessing the VBE Functions**

- As funções VBE são chamas usando a interface da BIOS (ou seja, via INT 0x10). Ao chamar, o registo *AH* deve ser posto a *0x4F,* distinguindo-se assim da norma da VGA BIOS. A função chamada é distinguível através do registo AL.

- Neste lab usa se a função *sys\_int86()* para invocar uma função da VBE. Tem como parâmetro de entrada um pointer para uma struct do tipo reg86\_t, em que:

* o campo ax deve ter o número 0x4F mais a função da VBE que queremos utilizar;
* o campo bx que deve ter o modo VBE para que queremos mudar (falado mais à frente) com o bit 14 garantidamente a 1;
* o campo intno que deve ser igual a 0x10

Exemplo:

**reg86\_t r;**

**r.ax = 0x4F02; // VBE call, function 02 -- set VBE mode**

**r.bx = 1<<14|0x105; // set bit 14: linear framebuffer**

**r.intno = 0x10;**

**if( sys\_int86(&r) != OK ) {**

**printf("set\_vbe\_mode: sys\_int86() failed \n");**

**return 1;**

**}**

**Funções VBE:**

- Existem várias funções da VBE:

* **0x00,**  que retorna a informação do controlador
* **0x01,** que retorna a informação do modo gráfico atual
* **0x02,** que muda o modo gráfico para outro especificado

**3.2 Setting the Graphics Mode**

- A norma VBE define vários modos de operação com características diferentes

|  |  |  |  |
| --- | --- | --- | --- |
| **Mode** | **Screen Resolution** | **Color Model** | **Bits per pixel (R:G:B)** |
| 0x105 | 1024x768 | Indexed | 8 |
| 0x110 | 640x480 | Direct color | 15((1:)5:5:5) |
| 0x115 | 800x600 | Direct color | 24 (8:8:8) |
| 0x11A | 1280x1024 | Direct color | 16 (5:6:5) |
| 0x14C | 1152x864 | Direct color | 32 ((8:)8:8:8) |

* A **Screen Resolution** define quantos pixéis existem na tela (no caso do modo 0x105, significa que existem 1024 pixéis na horizontal e 768 pixéis na vertical)
* A **Color Model** define como se guarda a cor (falado mais à frente
* O **Bits per pixel** define quantos bits existem num pixel

**3.4 Returning to Text Mode**

- Para retornar ao modo texto do Minix, deve-se usar a função vg\_exit()

**int vg\_exit() {**

**reg86\_t reg86;**

**reg86.intno = 0x10;**

**reg86.ah = 0x00;**

**reg86.al = 0x03;**

**if( sys\_int86(&reg86) != OK ) {**

**printf("vg\_exit(): sys\_int86() failed \n");**

**return 1;**

**}**

**return 0;**

**Mapear Memória Física em Memória Virtual**

- O acesso a memória física é feito usando a função *vm\_map\_phys()*. Esta função retorna o endereço de memoria virtual em que a região está mapeada.

void \*vm\_map\_phys(endpoint\_t who, void \*phaddr, size\_t len);

- Antes de invocar a função em cima declarada, é necessário invocar a sys\_privctl() para ter permissão para mapear memória física.

int sys\_privctl(endpoint\_t proc\_ep, int request, void \*p)

**Exemplo:**

mr.mr\_base = (phys\_bytes) vram\_base;

mr.mr\_limit = mr.mr\_base + vram\_size;

if( OK != (r = sys\_privctl(SELF, SYS\_PRIV\_ADD\_MEM, &mr)))

panic("sys\_privctl (ADD\_MEM) failed: %d\n", r);

/\* Map memory \*/

video\_mem = vm\_map\_phys(SELF, (void \*)mr.mr\_base, vram\_size);

if(video\_mem == MAP\_FAILED)

panic("couldn't map video memory");

**XPM**

- Um XPM (X Pixmap) é um formato de imagem que permite representar uma imagem como um array de valores para os pixéis

static char \*pic1[] = {

"32 13 4", /\* number of columns and rows, in pixels, and colors ". 0", /\* ’.’ denotes color value 0 \*/

"x 2", /\* ’x’ denotes color value 2 \*/

"o 14", /\* .. and so on \*/

"+ 4",

"................................", /\* the map \*/

"..............xxx...............",

"............xxxxxxx.............",

".........xxxxxx+xxxxxx..........",

"......xxxxxxx+++++xxxxxxx.......",

"....xxxxxxx+++++++++xxxxxxx.....",

"....xxxxxxx+++++++++xxxxxxx.....",

"......xxxxxxx+++++xxxxxxx.......",

".........xxxxxx+xxxxxx..........",

"..........ooxxxxxxxoo...........",

".......ooo...........ooo........",

".....ooo...............ooo......",

"...ooo...................ooo...."

};

- Para fazer load de um XPM, usa