Resumo de Estudo

Capítulo 2 — RAM e Vídeo

Carolina Rocio, Enzo Ofrante, Isabela Pinheiro, Kauã Araujo e Lucas Cunha

6 de abril de 2026

Resumo

Este material apresenta um resumo estruturado e aprofundado sobre os tópicos de RAM e vídeo abordados no Capítulo 2 do conteúdo estudado. O objetivo é oferecer uma base sólida para compreensão da matéria, permitindo não apenas memorização, mas também entendimento conceitual suficiente para responder questões teóricas e discursivas em prova. O foco principal está nas limitações do modo real, no problema do endereçamento segmentado, no uso de memória estendida por meio de EMS e XMS, na arquitetura VGA, nos modos gráficos 12h e 13h, e na importância do double buffering para evitar tearing.

Sumário

1 Visão geral da matéria

Este conteúdo mostra como os computadores pessoais do início dos anos 1990, apesar de terem evoluído em hardware, ainda eram muito limitados por decisões antigas de arquitetura e por exigências de compatibilidade. O capítulo trabalha principalmente dois blocos: a memória RAM e o sistema de vídeo.

Na parte de memória, o ponto central é entender que os processadores já tinham capacidade de trabalhar melhor, especialmente no modo protegido, mas os programas ainda eram fortemente limitados pelo MS-DOS, que operava em modo real. Isso restringia o uso de memória, complicava o endereçamento e dificultava a programação.

Na parte de vídeo, o foco está no padrão VGA. Embora ele fosse o padrão dominante da época, sua arquitetura era complexa e difícil de programar com bom desempenho. O texto mostra como limitações físicas da memória e do monitor influenciavam diretamente a maneira como jogos e aplicações gráficas eram construídos.

A grande ideia da matéria é que o programador não podia pensar apenas em lógica de software: ele precisava conhecer profundamente o hardware para conseguir desempenho aceitável.

2 RAM

2.1 Evolução do x86

Os primeiros processadores da família Intel x86 foram pensados em uma época em que a memória era cara. Por isso, modelos como o 8080 e o 8086 possuíam:

- registradores de 16 bits;
- barramento de endereços de 20 bits;
- capacidade de endereçar até 1 MiB de RAM.

Mais tarde surgiram o 286 e o 386, trazendo melhorias importantes:

- modo protegido;
- barramento maior;
- mais memória acessível;
- registradores de 32 bits;
- uso de MMU para proteção de memória.

Mesmo assim, isso não significava que os programas passavam automaticamente a usar esses recursos. A compatibilidade com programas antigos mantinha o sistema preso ao modo real.

2.2 Modo real e modo protegido

Essa diferença é essencial para a prova.

2.2.1 Modo real

O modo real reproduzia o comportamento antigo do 8086. Suas características principais eram:

- registradores de 16 bits;
- barramento de endereços de 20 bits;
- limite de 1 MiB de memória endereçável;
- endereçamento segmentado.

2.2.2 Modo protegido

O modo protegido representava uma evolução importante:

- registradores de 32 bits;
- acesso a mais memória;
- memória linear;
- proteção por MMU;
- ambiente mais moderno e robusto.

A ideia mais importante é:

> Modo real é antigo, limitado e segmentado; modo protegido é mais moderno, mais poderoso e melhor organizado.

2.3 Por que o DOS limitava a máquina

Apesar de o hardware já ter evoluído, o MS-DOS priorizava compatibilidade com programas antigos. Como muitos programas haviam sido escritos para modo real, o DOS continuava operando dessa forma.

Na prática, isso significava que programadores dos anos 90 usavam CPUs modernas como se ainda estivessem programando para máquinas muito antigas. Eles eram forçados a

abrir mão de recursos importantes e trabalhar com:

- registradores de 16 bits;
- até 1 MiB de RAM;
- endereçamento segmentado;
- várias dificuldades extras na linguagem C e na organização da memória.

Esse é um ponto forte para prova: o problema não era apenas o processador, mas o conjunto **hardware + sistema operacional + compatibilidade retroativa**.

2.4 O limite de 1 MiB de RAM

No modo real, apenas 1 MiB podia ser endereçado. Isso não quer dizer que o programa podia usar 1 MiB inteiro. Parte dessa memória era ocupada por:

- tabela de vetores de interrupção;
- dados da BIOS;
- componentes do sistema;
- área superior reservada para dispositivos.

Por isso, sobravam para o programa apenas cerca de 640 KiB, chamados de **Memória Convencional**.

Acima disso, existia a **UMA (Upper Memory Area)**, usada por BIOS, placa de vídeo, placa de som e outros dispositivos mapeados em memória.

Portanto, o ponto correto para lembrar é:

> Mesmo com 1 MiB endereçável, o programa geralmente só tinha acesso real a aproximadamente 640 KiB.

2.5 Endereçamento segmentado

Esse é um dos tópicos mais importantes de toda a parte de RAM.

Como os registradores tinham 16 bits, mas o endereço completo exigia 20 bits, a Intel adotou um sistema em que o endereço era formado pela combinação de:

- um segmento;
- um offset.

Isso recebia o nome de **endereçamento segmentado de 16 bits**.

2.5.1 Tipos de ponteiro

- **Near pointer**: tinha 16 bits, era mais rápido, mas limitado ao segmento atual.
- **Far pointer**: permitia acessar outras regiões, mas era mais lento e mais complicado.

2.5.2 Problemas causados por esse modelo

Esse sistema gerava vários problemas:

1. A linguagem C precisou ser adaptada com palavras-chave como `near` e `far`.
2. O `malloc` comum só conseguia alocar até 64 KiB.
3. Era possível ter ponteiros diferentes apontando para o mesmo endereço físico.
4. Comparações entre ponteiros podiam falhar mesmo quando eles levavam ao mesmo lugar.
5. A aritmética de ponteiros se tornava perigosa em blocos maiores que 64 KiB.

Ou seja, não era apenas uma dificuldade teórica: isso atrapalhava diretamente a programação prática.

2.6 Memória estendida

As máquinas de 1992 normalmente tinham mais que 1 MiB de RAM física. Muitas tinham 2 MiB ou 4 MiB. O problema era que o modo real não conseguia acessar naturalmente essa memória adicional. Essa parte acima de 1 MiB era chamada de **memória estendida**.

Para usá-la, era necessário instalar drivers especiais. Os dois padrões principais eram:

- EMS — Expanded Memory Specification;
- XMS — eXtended Memory Specification.

Sem esses drivers, o usuário podia ter RAM instalada e ainda assim o programa dizer que não havia memória suficiente.

2.7 XMS e EMS

2.7.1 XMS

O XMS funcionava de forma mais intuitiva. Ele lembrava o modelo de alocar e mover dados, parecido com `malloc/free`. Mas o ponto principal era que os dados precisavam ser **copiados** entre a memória estendida e a memória convencional.

2.7.2 EMS

O EMS trabalhava com uma lógica diferente. Em vez de copiar os dados, ele criava uma janela chamada **Page Frame**, permitindo mapear páginas de memória para dentro dessa janela. Assim, a memória podia ser acessada por mapeamento.

2.7.3 Comparação

A diferença principal é:

- XMS trabalha mais com cópia;
- EMS trabalha com mapeamento.

Por isso, o EMS era consideravelmente mais rápido. Esse detalhe foi importante no gerenciamento de memória de jogos da época, como *Wolfenstein 3D*.

2.8 Por que esse sistema era tão problemático

O conjunto formado por arquitetura antiga, compatibilidade com o DOS, limite de memória e endereçamento segmentado tornava o ambiente de programação muito difícil.

Para contornar isso, surgiam técnicas como **overlays**, nas quais apenas partes do programa eram carregadas na memória, e outras partes eram buscadas do disco quando necessário. Isso acontecia porque o código executável nem sempre cabia inteiro na memória disponível.

Portanto, a parte de RAM mostra um cenário em que o programador precisava fazer muito esforço apenas para conseguir usar a memória de forma eficiente.

3 Vídeo

3.1 Monitores CRT e o contexto da época

Os PCs da época usavam monitores CRT, que eram grandes, pesados e analógicos. A maioria tinha proporção 4:3. Como o computador operava de forma digital, era necessário um sistema intermediário para converter os dados do computador em sinal de vídeo compreensível pelo monitor.

Essa função era realizada pelos **adaptadores de vídeo**.

3.2 Evolução dos adaptadores

O conteúdo apresenta uma sequência histórica importante:

- MDA;
- CGA;
- EGA;
- VGA.

Cada geração adicionava recursos, mas preservava compatibilidade com as anteriores. Em 1991, o padrão mais importante era o VGA.

A vantagem disso era a padronização. A desvantagem era que todos ficavam presos às limitações da arquitetura VGA.

3.3 Arquitetura do VGA

A arquitetura VGA pode ser dividida em três grandes partes:

1. controladores de acesso à memória de vídeo;
2. VRAM;
3. controladores de saída para o monitor.

O mais importante é entender a VRAM. Em vez de um único bloco linear de 256 KiB, o VGA usava **quatro bancos de 64 KiB**.

3.3.1 Por que quatro bancos?

A RAM da época tinha latência alta. Se o sistema dependesse de apenas um banco, não haveria velocidade suficiente para alimentar o monitor na frequência exigida. Com quatro bancos lidos em paralelo, a taxa de transferência ficava adequada.

Assim, a arquitetura do VGA foi moldada pelas limitações físicas do hardware.

3.4 Arquitetura planar

A memória de vídeo do VGA não era organizada de forma simples e linear. Ela seguia uma organização **planar**, isto é, distribuída em planos ou bancos.

Isso tornava a programação difícil porque:

- não era trivial saber onde cada pixel deveria ser escrito;
- era necessário coordenar múltiplos planos;
- havia muitos registradores internos mal documentados.

Esse é um ponto conceitual forte: o VGA não era difícil por escolha estética, mas porque a solução técnica da época exigia esse tipo de organização.

3.5 Modos VGA

Os modos mais importantes para estudo são:

- **Modo 12h**: 640x480, 16 cores;
- **Modo 13h**: 320x200, 256 cores.

Esses dois modos apareciam com frequência em programação de jogos porque representavam os dois principais caminhos disponíveis:

- maior resolução com menos cores;
- menor resolução com mais cores.

3.6 Mapeamento da memória de vídeo

No VGA, parte do espaço de endereçamento da RAM era usada para mapear a VRAM. No modo 13h, por exemplo, a memória de vídeo ficava mapeada em um intervalo específico.

O problema é que a VRAM total podia chegar a 256 KiB, mas o espaço visível de uma vez não era suficiente para isso. Para resolver essa limitação, usava-se **bank switching**, ou seja, alternância entre bancos da memória de vídeo.

3.7 Modo 12h

O modo 12h oferecia:

- resolução de 640x480;
- 16 cores;
- pixels quadrados.

A vantagem dos pixels quadrados era importante porque a imagem não sofria distorção no CRT com proporção 4:3.

No entanto, esse modo tinha várias desvantagens:

- alta resolução, o que gerava mais cálculos;
- apenas 16 cores, o que limitava a qualidade visual;
- ausência prática de double buffer;
- escrita mais complicada por causa da organização em planos.

Em resumo, o modo 12h tinha uma geometria boa, mas era pesado e visualmente limitado.

3.8 Modo 13h

O modo 13h era muito mais atraente para jogos porque oferecia:

- 320x200;
- 256 cores;
- programação mais simples;
- aparência de framebuffer linear.

Isso era viabilizado por um mecanismo de hardware chamado **Chain-4**. Esse chip usava os bits menos significativos do endereço para decidir automaticamente qual banco da VRAM seria acessado.

Na prática, isso facilitava muito o trabalho do programador, pois escondia parte da complexidade do sistema.

3.8.1 Problemas do modo 13h

Apesar de mais amigável, ele também tinha limitações:

- desperdiçava parte da memória;
- não resolvia bem a questão do double buffer;
- a proporção 320x200 não combinava com a tela 4:3;
- a imagem era esticada, fazendo círculos parecerem elipses.

Ou seja, o modo 13h era melhor para desenhar e para exibir mais cores, mas tinha distorção geométrica.

3.9 Configuração do modo 13h

Um detalhe importante é que ativar o modo 13h pela BIOS era simples. Bastava usar uma interrupção de vídeo, como no exemplo clássico:

```
mov ax, 0x13
int 0x10
```

Depois disso, o programa podia escrever diretamente na memória de vídeo mapeada. Esse detalhe mostra como a BIOS ajudava a esconder a complexidade de configuração dos muitos registradores internos do VGA.

3.10 Double buffering e tearing

Esse é um dos conceitos mais importantes de vídeo.

3.10.1 Tearing

O **tearing** acontece quando a CPU altera o framebuffer enquanto o monitor ainda está exibindo esse mesmo framebuffer. Com isso, parte da imagem exibida pertence a um frame antigo e parte pertence a um frame novo. O resultado parece uma imagem rasgada.

3.10.2 Double buffering

O **double buffering** resolve esse problema usando dois buffers:

- um buffer está sendo exibido;
- o outro buffer está sendo desenhado.

Quando o novo quadro fica pronto, os buffers trocam de papel. Assim, a CPU não interfere na imagem que ainda está sendo mostrada pelo monitor.

A ideia principal para prova é:

> Um único buffer pode causar tearing; dois buffers tornam a animação mais suave e estável.

4 Síntese final da matéria

O capítulo mostra que a programação para PCs do início dos anos 90 era profundamente influenciada pelas limitações do hardware e pelo peso da compatibilidade com o passado.

Na parte de RAM, os conceitos mais importantes são:

- diferença entre modo real e modo protegido;
- limitação do DOS;
- limite de 1 MiB de memória endereçável;
- apenas cerca de 640 KiB livres para programas;
- endereçamento segmentado;
- ponteiros near e far;
- uso de XMS e EMS;
- vantagem de desempenho do EMS.

Na parte de vídeo, os conceitos principais são:

- evolução dos adaptadores até o VGA;
- VRAM organizada em quatro bancos;
- arquitetura planar;
- diferenças entre modo 12h e modo 13h;
- Chain-4;
- bank switching;
- double buffering;
- tearing.

A principal conclusão é que o software da época dependia fortemente do entendimento do hardware. Quem programava precisava conhecer não apenas lógica e algoritmos, mas também a organização da memória, a forma como os registradores funcionavam e os limites físicos do vídeo e da RAM.