CENTRO UNIVERSITÁRIO FEEVALE

ANDRÉ WAGNER

CRIAÇÃO DE UM KIT DE DESENVOLVIMENTO DE SOFTWARE PARA PROGRAMAÇÃO MODULAR DE EMULADORES

NOVO HAMBURGO, 2006

ANDRÉ WAGNER

CRIAÇÃO DE UM KIT DE DESENVOLVIMENTO DE SOFTWARE PARA PROGRAMAÇÃO MODULAR DE EMULADORES

Centro Universitário Feevale
Instituto de Ciências Exatas e Tecnológicas
Curso de Ciência da Computação
Trabalho de Conclusão de Curso

Professor orientador: Delfim Luiz Torok

Novo Hamburgo, dezembro de 2006

DEDICATÓRIA

RESUMO

ABSTRACT

LISTA DE ABREVIATURAS E SIGLAS

ALU	*Aritmethic Logic Unit*	Unidade Lógica Aritmética
BIOS	*Basic Input/Output System*	Sistema Básico de Entradas e Saídas
CISC	*Complex Instruction Set Computer*	Computador com um Conjunto de Instruções Complexo
CPU	*Central Processing Unit*	Unidade Central de Processamento
EPROM	*Eraseble Programmable Read-Only Memory*	Memória Somente Leitura Programável Apagável
GPU	*Graphics Processing Unit*	Unidade de Processamento Gráfico
HSYNC	*Horizontal Synchronization*	Sincronização Vertical
INT	*Interruption*	Interrupção
IP	*Instruction Pointer*	Ponteiro de Instrução
JVM	*Java Virtual Machine*	Máquina Virtual Java
MSX	*MicroSoft eXtended (ou) Machines with Software eXchangeability*	Microsoft Estendido (ou) Máquinas com Compatibilidade de Software
NTSC	*National Television System Committee*	Comitê Nacional de Sistema Televisivo
PAL	*Phase Alternation by Line*	Alternância de Fase por Linha
PC	*Personal Computer*	Computador Pessoal
PC	*Program Counter*	Contador de Programa
PROM	*Programmable Read-Only Memory*	Memória Somente Leitura Programável
RAM	*Random Access Memory*	Memória de Acesso Randômico
RGB	*Red Green Blue*	Vermelho Verde Azul
RISC	*Reduced Instruction Set Computer*	Computador com um Conjunto de Instruções Reduzido
SP	*Stack Pointer*	Ponteiro de pilha
TIA	*Television Interface Adapter*	Adaptador de Interface de Televisão
UAL		Unidade Aritmética Lógica
UCP		Unidade Central de Processamento
VBLANK	*Vertical Blank*	Área Vazia Vertical
VDP	*Video Display Processor*	Processador de Exibição de Imagens
VPU	*Video Processing Unit*	Unidade de Processamento de Vídeo
VSYNC	*Vertical Synchronization*	Sincronização Vertical
WINE	*Wine Is Not a Emulator*	Wine Não é um Emulador

SUMÁRIO

INTRODUÇÃO

1 CONCEITOS INTRODUTÓRIOS SOBRE EMULADORES

Este capítulo objetiva introduzir o assunto de emuladores, definindo o que são, quais são os seus tipos básicos e a sua filosofia de funcionamento, bem como situando os objetivos propostos por este trabalho dentro do universo dos tipos e das técnicas de emulação.

1.1 Emuladores

Um emulador é um programa de computador que simula o comportamento de uma arquitetura computacional[1], de modo que um software escrito para uma plataforma possa ser executado em outra. A Sociedade Britânica de Computação define a emulação como "[...] uma forma precisa de simulação que imita exatamente o comportamento ou as circunstâncias que se estão simulando. Um emulador permite que um tipo de computador opere como se fosse um tipo diferente de computador." [tradução nossa] (BRITISH, 2002, p. 30-31).

Um emulador assemelha-se bastante a uma máquina virtual, como a Máquina Virtual Java (JVM, do inglês *Java Virtual Machine*). Neste caso, o código é escrito na linguagem Java pelo programador, e compilado para um *bytecode*[2]. Este bytecode – semelhante a um código binário executável – é então interpretado por uma máquina virtual, como o JVM. Segundo Lindholm (1999, p. ?), "a Máquina Virtual Java é um computador abstrato. Como um computador real, possui um conjunto de instruções e manipula várias áreas de memória em tempo de execução." [tradução nossa]

A diferença essencial entre um emulador e uma máquina virtual está no fato da máquina virtual interpretar código escrito para uma máquina abstrata, enquanto um emulador procura interpretar código escrito para uma máquina real, geralmente diferente daquela na qual o emula-

1 O conjunto de tipos de dados, operações e características de um computador. Alguns autores incluem o sistema operacional junto na arquitetura (fazendo de um PC com Windows uma arquitetura diferente de um PC com Linux). Neste trabalho, a definição refere-se exclusivamente à arquitetura de hardware de um computador.

2 Conjunto de instruções em forma binária, interpretáveis através de uma máquina virtual. Do inglês "código em bytes".

dor está sendo executado. No caso máquina virtual, o código foi escrito especificamente para ela, enquanto no caso do emulador, o código não foi escrito para o emulador, mas para outra arquitetura computacional.

Um exemplo de emulador é o fMSX. O fMSX é um software que simula o funcionamento de um computador MSX[3] em uma série de sistemas operacionais diferentes, como Windows, MS-DOS, MacOS, Linux, etc. Desta forma, é possível usar um computador PC[4] com Windows (ou MS-DOS, ou outros sistemas operacionais) e, ainda assim, executar qualquer programa escrito para o MSX (FAYZULLIN, 2006).

A performance de um emulador é muito importante, pois é necessário que o usuário tenha a impressão de que está usando uma máquina real. Para isso, o emulador precisa manter a sincronização, de modo a não ser executado nem muito rápida nem muito lentamente (DEL BARRIO, 2001, p. 11).

1.2 Níveis de Abstração

Abstração é o processo de reduzir a quantidade de informação necessária para um determinado conceito, com o objetivo de eliminar os detalhes irrelevantes e reter somente a informação relevante para um determinado propósito. Esta idéia é usada para gerenciar a complexidade de sistemas computacionais. Segundo Keller (1997, p. 1),

> estes sistemas geralmente consistem de milhões de pequenos componentes (bytes de memória, comandos de programa, portas lógicas, etc.). Tratar todos os componentes como um simples monolito é quase intelectualmente impossível. Portanto é comum, ao invés disso, ver o sistema como sendo composto por alguns poucos componentes interativos, cada qual compreendido em termos de *seus* componentes, e assim por diante, até que o nível mais básico seja atingido. [tradução nossa] [grifo do autor]

Tanenbaum (1990) define os computadores como tendo, em geral, seis níveis de abstração, partindo do nível mais baixo (hardware) e indo para o nível mais alto (aplicações), segundo mostrado na Figura 1.

---

3 Computador de arquitetura aberta criado por uma parceria entre Microsoft e ASCII, de modo a oferecer uma arquitetura padrão. Usava um microprocessador Zilog Z80. Foi muito popular no Brasil nos anos 80, onde era fabricado pela Gradiente (com o nome de Expert) e pela Sharp (com o nome de Hotbit).

4 Computador pessoal padrão IBM usando microprocessador da série x86. Do inglês *Personal Computer*.

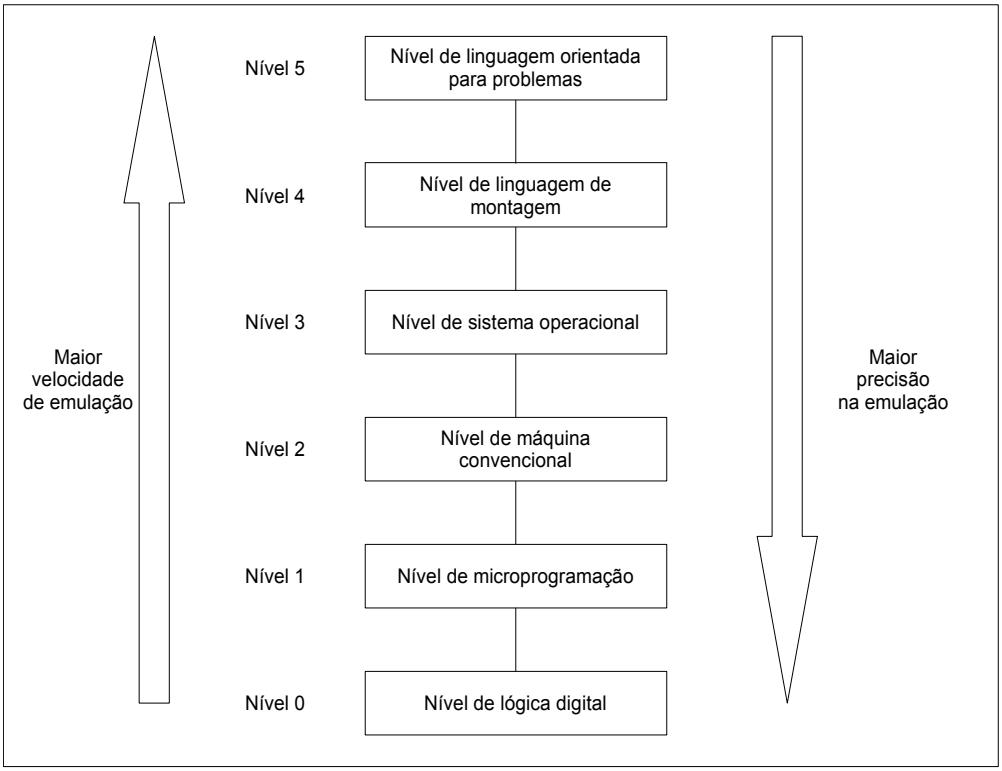


Figura 1 - Os seis níveis de abstração definidos por Tanenbaum.

O nível mais baixo é o **nível de lógica digital**, que é o verdadeiro hardware da máquina. Este nível inclui especialmente as portas lógicas, que são dispositivos digitais que executam operações simples de lógica booleana[5], como as operações E e OU (TANENBAUM, 1990).

Os dois níveis seguintes são o **nível de microprogramação** e o **nível de máquina convencional**. Embora ambos pareçam bastante semelhantes, o nível de máquina convencional é aquele na qual as instruções *assembly*[6] são executadas; portanto, quando o fabricante de um determinado processador distribui um manual de referência de linguagem *assembly*, este manual refere-se ao nível 2, e não ao nível 1. O nível 1, de microprogramação, refere-se a como estas instruções são implementadas dentro do processador (TANENBAUM, 1990).

O nível seguinte é o **nível de sistema operacional**. Em um sistema operacional, a maior parte das instruções executadas são idênticas às do nível dois, porém um novo conjunto de funcionalidades é acrescentado pelo sistema operacional. Segundo Tanenbaum, "algumas ins-

5 Sistema lógico baseado em valores digitais (VERDADEIRO e FALSO), criada por George Bool no século XIX, e atualmente muito usada na área da eletrônica e da computação.

6 Linguagem de montagem, onde as instruções numéricas enviadas para o processador (linguagem de máquina) são representadas por abreviaturas, de modo a facilitar o uso por um programador. Exemplos de instruções *assembly* comuns são MOV (mover), ADD (somar) e INT (gerar interrupção).

truções de nível 3 são interpretadas pelo sistema operacional e outras são interpretadas diretamente pelo microprograma." (TANENBAUM, 1990, p. 5). As novas funcionalidades acrescentadas pelo sistema operacional são executadas através de chamadas de sistema[7].

O **nível de linguagem de montagem** e o **nível de linguagem orientada para problemas** são bastante diferentes dos outros níveis. Eles são voltados para programadores de aplicação que têm problemas específicos a serem solucionados. Estes níveis definem, respectivamente, os dois passos pela qual uma aplicação em código-fonte passa até se transformar em um arquivo binário executável: a montagem e a compilação. Um código-fonte em uma linguagem de alto nível (como C, BASIC ou Pascal) é **compilado** para um código-fonte *assembly*, e o código-fonte *assembly* é **montado** em um arquivo binário executável (TANENBAUM, 1990).

1.3 Emuladores e Níveis de Abstração

Existem diferentes tipos de emuladores que simulam outras arquiteturas, e que variam de acordo com o nível de abstração que emulam. Quanto mais alto o nível de abstração a ser emulado, maior será a velocidade, e menor a precisão. O oposto também é verdadeiro - quanto mais exata a emulação, menor será a performance.

Simuladores são programas que buscam imitar o funcionamento de uma arquitetura com precisão absoluta – ou seja, emulam o nível 0. Geralmente são usados por engenheiros com o objetivo de testar novos tipos de hardware ou software, ou então para compreender como um sistema age internamente (um exemplo seria compreender como um determinado sistema operacional está usando a memória *cache*[8]). Devido à sua precisão, um sistema simulado geralmente não consegue atingir uma velocidade equivalente à do hardware original (DEL BARRIO, 2001, p. 12).

Um **emulador** funciona de forma semelhante a um simulador, mas busca obter uma velocidade compatível à do hardware original. Para isto, um emulador muitas vezes precisa deixar um pouco da precisão de lado para obter um desempenho superior (este fator será discutido com mais profundidade na Seção 3.4). Segundo Del Barrio (2001, p. 16),

> Outro aspecto da emulação que deve ser levado em conta é o nível de precisão que o emulador necessita. O tipo de emulador ao qual nos referimos não necessita ser preciso em um nível muito baixo (por exemplo, no funcionamento interno do microprocessador, nas funções de *cache* ou no tráfego de barramento) porque a intenção não é analisar a performance ou simular o computador. Queremos emular o comportamento externo dos *videogames* (ou aplicativos) executados no emulador o mais próximo

7 Mecanismo utilizado por um aplicativo para requisitar um serviço do sistema operacional, geralmente através de uma interrupção de software.

8 Memória de acesso muito rápido, onde uma cópia dos dados mais utilizados da memória RAM são armazenados.

> possível do computador real. A intenção é que o emulador soe e se pareça o máximo possível com o sistema real. Isto significa que, às vezes, a precisão precisa ser sacrificada pelo desempenho, porque uma das coisas mais importantes a serem emuladas em um emulador é a "sensação de tempo", ou seja, o emulador precisa ser executado exatamente à mesma velocidade do sistema real. [tradução nossa]

Os emuladores estão no nível 2 de abstração definido por Tanenbaum, conforme ilustra a Figura 1, denominado como nível de máquina convencional.

Um **emulador de alto nível** é um tipo especial, pois é construído com o objetivo de obter um desempenho superior. Nele, as chamadas de sistema dos aplicativos são capturadas e, ao invés de serem interpretadas, são executadas por instruções que estão dentro do próprio emulador. Isto significa que um emulador de alto nível abrange também o nível 3. A desvantagem deste método é que ele é pouco preciso, e para cada software da plataforma que está sendo emulada, é necessário testar e adaptar o emulador. Um exemplo deste tipo de emulador é o UltraHLE, que emula um videogame do tipo Nintendo 64 (ULTRAHLE, 2006).

Um **virtualizador** é um tipo de emulador que implementa apenas a própria arquitetura onde está sendo executado. Ele faz isso emulando apenas os componentes e executando o código binário diretamente no processador, atingindo assim uma velocidade quase nativa. É usado, por exemplo, para executar um sistema operacional diferente do que se está sendo executado, mas que rode na mesma arquitetura (por exemplo, Windows e Linux). Um exemplo de virtualizador é o VMWare (VMWARE, 2006).

O virtualizador, assim como o emulador, abrange o nível 2, sendo que a diferença entre os dois está no fato de que o emulador interpretar as instruções do processador, enquanto o virtualizador as executa diretamente.

Por último, existem ainda os **simuladores de sistema operacional**. Neste caso, eles agem como virtualizadores, mas ao invés de emular os componentes de hardware, eles capturam as chamadas de sistema e as implementam em uma plataforma diferente. Exemplos deste tipo de software são o *Cooperative Linux* (que permite usar programas de Linux dentro do Windows) e o WINE[9] (que permite usar programas de Windows no Linux) (ALONI, 2004).

Levando em conta todos estes tipos de programas simuladores, a construção de emuladores que se enquadram no nível 2 (segundo definido por Tanenbaum) será o foco principal deste trabalho.

9 "*Wine* não é um Emulador" - programa que permite executar programas de Windows no Linux, emulando as chamadas de sistema à API do Windows (do inglês *Wine Is Not a Emulator*)

1.4 Filosofia de Funcionamento

Segundo Von Neumann, os computadores são formados por um conjunto de, no mínimo, cinco componentes: uma unidade de memória (hoje geralmente conhecida por memória RAM[10]), uma unidade de entrada (um teclado, por exemplo), uma unidade de saída (um monitor de vídeo, por exemplo), uma unidade de controle e uma unidade de lógica aritmética (hoje unificadas em um microprocessador), todos conectados através de um barramento (MURDOCCA, 2000).

Assim como um computador, um emulador funciona de forma modular, com os módulos do emulador tendo função semelhante àqueles do computador. Desta forma, existe um módulo que emula um microprocessador, um módulo que emula a memória, um módulo que emula os gráficos, e assim por diante. Estes módulos são unificados através de um laço[11], que de forma simplificada para uma máquina monoprocessada poderia ser o da Figura 2.

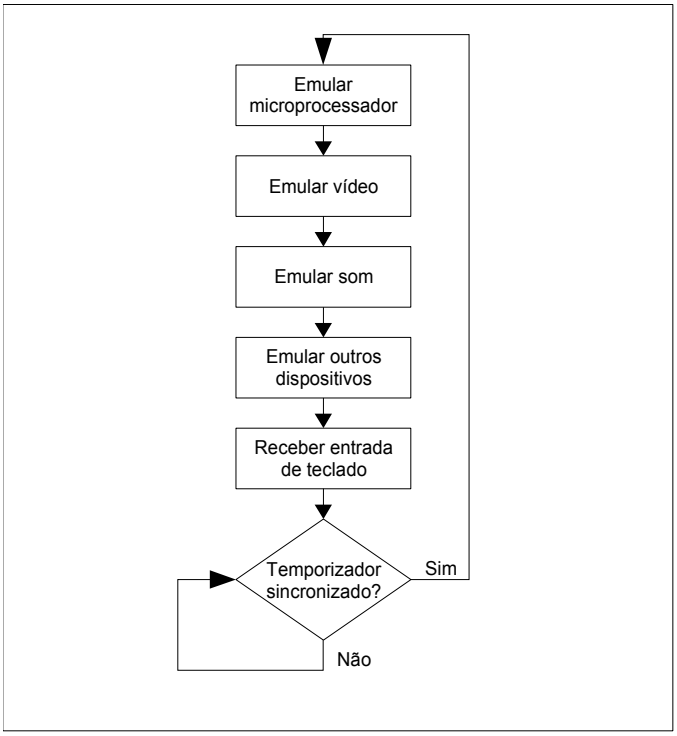


Figura 2 - Laço de um emulador simples

A emulação de um **microprocessador** é diferente da emulação dos outros tipos de dispositivos. O microprocessador busca dados da memória e executa as instruções equivalentes àqueles dados, retornando à memória o resultado da execução, ou então realizando alguma ou-

10 Memória de Acesso Randômico, significando uma memória que pode ser acessada em qualquer posição, ao contrário dos outros tipos de memória usados antigamente, como fitas (do inglês *Random Access Memory*)

11 Um conjunto de instruções de um programa que é definido apenas uma vez, mas é executado repetidamente. Também conhecido pela palavra inglesa *loop*.

tra operação (por exemplo, comunicando-se com algum componente do sistema). A execução do microprocessador será interrompida com uma certa freqüência para tratar eventos de outros componentes, num processo chamado interrupção. (DEL BARRIO, 2001). A emulação de microprocessadores será discutida mais profundamente na sessão 3.2.

É necessário que haja uma sincronia entre o microprocessador e os outros componentes. Esta sincronia é atingida através de um sinal de ***clock***, que é um sinal digital usado para coordenar as ações de todos os circuitos (WIKIPEDIA, 2006a). No emulador, diferentemente da máquina real, todos os componentes são guiados pelo microprocessador – ele é o responsável por produzir a informação de quantos ciclos de *clock* foram gerados com a última instrução, e esta informação é passada aos outros componentes no momento de sua emulação (BORIS, 1999).

Outros componentes internos do computador também devem ser emulados. Geralmente, eles se comunicam com o microprocessador e uns com os outros através de **áreas de memória** (onde cada componente tem uma área de memória reservada para si), de portas e de interrupções. Computadores modernos também usam um sistema de Acesso Direto à Memória (DMA, do inglês *Direct Memory Access*); segundo Murdocca (2000, p. 288), "um dispositivo com acesso direto à memória pode transferir dados diretamente da memória para para a mesma em vez de usar a CPU[12] como intermediário; portanto, pode diminuir a congestão do barramento do sistema."

Dos componentes internos do computador a serem emulados, geralmente o mais complexo e que utiliza mais de recursos de emulação é o **sistema gráfico**. De acordo com Del Barrio (2001, p. 107), "a emulação gráfica [...] ocupará mais de 50% do tempo da emulação, às vezes chegando até 80% ou 90%" [tradução nossa]. Assim, na emulação do sistema gráfico – e, algumas vezes, de outros componentes – se faz necessário sacrificar a exatidão da emulação em prol do desempenho.

O **sistema de entrada**, geralmente baseado em teclado e mouse (no caso de microcomputadores) ou em *joystick*[13] (no caso de videogames), é normalmente a parte mais simples da emulação. Neste caso, quando uma ação for realizada na plataforma onde o emulador está sendo executado (por exemplo, o pressionamento de uma tecla), a ação correspondente será realizada na plataforma emulada (por exemplo, será escrito um byte na memória ou uma interrupção será gerada).

12 Termo comumente usado para referir-se ao microprocessador. Do inglês *Central Processing Unit*, ou Unidade de Processamento Central.

13 Dispositivo de entrada usado em videogames, geralmente composto por uma manopla para o acionamento direcional e alguns botões.

1.5 Técnicas de Emulação

Segundo Laureano (2006), os emuladores podem ser baseados em hardware, software ou alguma combinação entre os dois. Emuladores baseados em hardware são aqueles que são totalmente implementados em um componente físico, como um *microchip*. Emuladores baseados em software são aqueles que são executados em um computador, totalmente separados da implementação de hardware e onde o software provê todos os recursos para a emulação. Existem ainda as soluções híbridas, onde parte da emulação é feita em hardware, e parte em software.

Os emuladores baseados em software, que são o foco principal deste trabalho, podem ser implementados usando quatro técnicas diferentes:

Usando a técnica de i**nterpretação,** cada instrução é interpretada por um módulo que emula o microprocessador. O emulador do microprocessador lê a próxima instrução da memória, busca seu código de execução em uma tabela de instruções e executa-a, realizando assim a função equivalente. Esta técnica tem a vantagem de ser o tipo de emulação mais completo – qualquer plataforma pode ser emulada em qualquer plataforma, e até mesmo um emulador pode ser executado dentro de outro emulador. Este é, entretanto, a técnica de implementação cuja a execução é, geralmente, a mais lenta (DEL BARRIO, 2001).

No caso da **tradução binária** estática, um emulador usando esta técnica funciona de forma semelhante a um compilador. Nele, um aplicativo escrito para uma arquitetura é compilado para código binário executável em outra arquitetura. Desta forma, um programa executável de uma arquitetura se transforma num programa executável de outra arquitetura. Este é o tipo mais rápido de emulação, porém tem sérias limitações, ele não consegue lidar com código auto-modificável, o que acontece em todos os computadores, quando um aplicativo é carregado de uma unidade de armazenamento (como um disquete) para a memória. Portanto, serve apenas para algumas plataformas específicas que não usam sistema operacional nem carregam programas para a memória RAM (como os videogames mais antigos). (DEL BARRIO, 2001)

Usando-se a técnica da **tradução binária dinâmica,** partes do código são compiladas (e recompiladas) durante a execução, sob demanda, de modo que o código gerado na memória venha a refletir as modificações que aconteceram durante a execução do emulador. Esta técnica é a mesma utilizada por máquinas virtuais do tipo *just-in-time* (JIT) (LAUREANO, 2006). Este tipo de emulador é mais rápido que o interpretador, mas mais lento que o tradutor binário estático, e seu grande problema é a extrema complexidade de implementação, devido à necessidade

de técnicas heurísticas[14] para detecção de blocos de código, detecção de memória modificada, compilação e otimização (DEL BARRIO, 2001).

A **virtualização** é a técnica mais rápida de todas, mas pode apenas ser usada em ambientes onde a arquitetura em que o emulador será executado e a arquitetura que será emulada são a mesma. Neste caso, apenas os componentes são emulados, mas o código é executado nativamente pelo microprocessador. Esta técnica é muito utilizada em computadores PC, de modo a permitir que dois sistemas operacionais sejam executados na mesma máquina, ou então para manter vários serviços rodando isoladamente em um mesmo computador (LAUREANO, 2006).

Este trabalho tratará dos emuladores que usam a técnica de interpretação, por ser a única das técnicas que permite uma boa modularização e o reaproveitamento dos componentes implementados, bem como a unificação das atividades padrão.

1.6 Benefícios do Uso de Emuladores

Os emuladores podem ser usados para diferentes fins:

- Facilitar o desenvolvimento em plataformas onde a programação é difícil ou impossível. Um exemplo é o desenvolvimento de jogos de videogame onde, em muitos casos, o desenvolvimento é feito em um computador e testado em um emulador. Outro exemplo são os dispositivos móveis (computadores portáteis e celulares), onde a pequena tela dificultaria a programação, mas com o uso de um emulador o programador não só tem a facilidade de poder testar seu desenvolvimento em uma tela maior, como também a agilidade de não precisar fazer um *upload* para o dispositivo a cada nova compilação;

- Permitir o uso de aplicativos legados (por exemplos, editores de texto ou jogos antigos), que eram utilizados em equipamentos que não estão mais disponíveis, ou aos quais o usuário não têm acesso;

- Facilitar o desenvolvimento de novos sistemas operacionais, permitindo que desenvolvedor tenha acesso a tudo que acontece dentro de cada dispositivo, e dando a ele a facilidade de testar seu sistema com diferentes configurações;

- Facilitar o desenvolvimento de programas para múltiplas plataformas, oferecendo ao desenvolvedor uma gama de arquiteturas nas quais ele pode testar seus programas sem a necessidade de copiá-los a outros computadores;

14 Métodos ou algoritmos exploratórios para definição de problemas em que as soluções são descobertas pela avaliação do progresso obtido na busca de um resultado final.

Além dos benefícios citados, Laureano (2006, p. 35) acrescenta ainda que os emuladores perimitem "Testar configurações e situações diferentes do mundo real, como, por exemplo, mais memória disponível ou a presença de outros dispositivos de E/S" e "auxiliar no ensino prático de sistemas operacionais e programação ao permitir a execução de vários sistemas para comparação no mesmo equipamento".

1.7 Considerações Finais

O aumento do número de arquiteturas disponíveis no mercado trouxe um aumento na demanda por novos tipos de emuladores. Existe uma vasta gama de técnicas de emulação, mas a implementação de qualquer uma delas é tarefa custosa, exigindo meses de desenvolvimento e teste, e a implementação de complexos algoritmos de modo a obter um desempenho equivalente à arquitetura original.

Devido à extensa quantidade de tipos e técnicas de emulação, este trabalho se baseará em emuladores do nível de máquina convencional (conforme discutido na seção 1.2), e que usem a interpretação como método de emulação, por ser o tipo que se encaixa de forma mais precisa com objetivos propostos neste trabalho.

2 ARQUITETURA DE HARDWARE PERTINENTE À CONSTRUÇÃO DE EMULADORES

Este capítulo discute a arquitetura interna de um computador. Visto que existe abundante literatura a respeito deste assunto, nesta seção serão tratados somente os assuntos pertinentes ao segundo nível de abstração, que é o nível convencional de máquina, segundo discutido na seção 1.2. Serão discutidos primeiramente cada um dos principais dispositivos que compõem um computador e, mais adiante, a forma como eles funcionam em conjunto dentro de uma arquitetura computacional.

As técnicas para a construção de emuladores dos dispositivos descritos neste capítulo serão discutidas no Capítulo 3.

2.1 O Processador

O processador (também conhecido como microprocessador, CPU ou UPC[15]) é o "cérebro" do computador. Tem como propósito executar programas armazenados na memória, buscando nela as instruções e executando-as, uma após a outra (TANENBAUM, 1990). Um processador comum é capaz de executar um grande número de instruções em um curto período de tempo: um Pentium 4[16], por exemplo, é capaz de executar 1 bilhão e 500 milhões de instruções por segundo. Já o 80386[17], bem mais modesto, opera a 5 milhões de instruções por segundo (GILHEANY, 2006).

Um processador é composto de:

- uma **unidade de controle**, responsável pela busca de instruções da memória principal e determinação de seus tipos;
- uma **unidade lógica aritmética** (ALU, do inglês *Arithmetic Logic Unit*), res-

15 Unidade Central de Processamento
16 Sétima geração de processadores do padrão x86 construída pela Intel, fabricado a partir de 2000.
17 Terceira geração de processadores do padrão x86 da Intel, fabricado a partir de 1986.

ponsável pela execução das operações aritméticas e lógicas com os dados;

- uma pequena **memória interna** de alta velocidade, composta por registradores, cada um com uma função definida (TANENBAUM, 1990).

2.1.1 Conjunto de Instruções de Máquina

Segundo Murdocca (2000, p. 99), "o **conjunto de instruções** é a coleção de instruções que um processador pode executar" [grifo do autor]. Em geral, estas instruções realizam operações bastante simples, como adições, subtrações, movimentação de dados dentro da memória, comparações entre posições da memória, etc. Esta simplicidade existe para aumentar a velocidade de execução, simplificar o *design* do processador e facilitar o trabalho do programador (PATTERSON, 2005). Uma vez que o processador executa milhões de operações por segundo, a complexidade da programação ficará nos níveis superiores de abstração, onde a programação geralmente é feita em linguagens de alto nível como C ou Pascal.

Para facilitar a programação, estas instruções são representadas através de mnemônicos. Num processador Intel da série x86, por exemplo, a instrução que gera uma interrupção de software é a instrução de número 0xCD[18]. Num código-fonte *assembly*, no entanto, este número será substituído pelo mnemônico INT (*Interruption*, do inglês "interrupção") (INTEL, 2006).

Nos processadores do tipo CISC[19], a instrução é composta por um byte. A maior parte das instruções, no entanto, recebem outros bytes que as acompanham. Estes bytes são chamados de **operandos**, e geralmente indicam a localização dos dados que serão manipulados durante a realização da operação (MONTEIRO, 1996). Num processador RISC[20], os operandos também estão presentes, embora não estejam em bytes separados – neste tipo de processador, todas as instruções (acrescidas de seus operandos) têm o mesmo número de bits (TANENBAUM, 1990).

Tanenbaum (1990) agrupa as instruções em sete tipos:

1. **instruções de transferência de dados**, que copiam dados de um lugar da memória para outro;

2. **operações diádicas**, que combinam dois operandos para produzir um resultado (por exemplo, uma instrução de soma);

3. **operações monádicas**, que têm um operando e produzem um resultado (por

18 Um conjunto de números ou letras precedido de "0x" significa que este é um número hexadecimal. Assim, 0xCD equivale ao número hexadecimal CD, ou ao número decimal 205.

19 Processador que pode executar várias operações de baixo nível em uma simples instrução. Do inglês *Complex Instruction Set Computer* (Computador com um Conjunto de Instruções Complexo).

20 Processador que favorece um conjunto de instruções mais simples que o CISC. Do inglês *Reduced Instruction Set Computer* (Computador com um Conjunto de Instruções Reduzido).

exemplo, uma instrução que incremente o valor de um determinado registrador em 1);

4. **comparações e desvios condicionais**, que comparam dois dados e transferem o controle do programa para uma posição especificada se eles forem iguais (ou diferentes, dependendo da instrução) – semelhante à instrução *if-then* (se-então) das linguagens de alto nível;

5. **instruções de chamada de procedimento**, que colocam o endereço atual na pilha[21] e transferem o controle do programa para outra posição – quando o procedimento termina, o endereço é desempilhado e o controle do programa é retornado à instrução seguinte de onde o procedimento foi chamado;

6. **controle de laço**, que permitem a execução repetida de partes de um programa, até que uma determinada condição seja satisfeita – de forma semelhante aos laços *do-while* (faça-enquanto) das linguagens de alto nível;

7. **entrada/saída**, através das quais é feita a comunicação com outros dispositivos. Neste grupo de instruções também entram as instruções de chamada de interrupção.

A Tabela 1 exemplifica a divisão acima, através de uma lista reduzida das operações do microprocessador MOS 6502[22].

Tabela 1 - Conjunto reduzido de instruções do MOS 6502, agrupadas por tipo.

Tipo de Instrução	Instrução	Descrição da Operação
Transferência de Dados	LDA	Carrega dados da memória para o acumulador
	PHA	Empilha dados do acumulador
Operações Diádicas	ADC	Soma dois operandos.
Operações Monádicas	INC	Incrementa uma posição da memória
Comparações e desvios condicionais	BEQ	Muda controle do programa se último resultado foi zero
	BPL	Muda controle do programa se último resultado foi negativo
Instruções de chamada de procedimento	JSR	Salta para sub-rotina
	RTS	Retorna de subrotina
Controle de laço		O 6502 usa as instruções de desvio condicional para o controle de laços.
Entrada/Saída	BRK	Gera uma interrupção de software

Fonte: JACOBS, 2002.

21 A pilha (*stack*) é uma área da memória que contém dados que são inseridos e lidos através de um algoritmo onde o último dado inserido é o primeiro a ser lido, semelhante a uma pilha de documentos.

22 Microprocessador de 8 bits criado pela MOS Techonology em 1975, extensivamente usado nos computadores pessoais e videogames nos anos 80.

2.1.2 Seqüência de Operação

O processador executa um ciclo repetitivo de operações, cada qual equivalente à execução de uma instrução. Este ciclo é chamado de **busca-decodificação-execução**, e é o centro da operação de todos os computadores (WEBER, 2000).

Tanenbaum (1990) divide a execução do ciclo em oito fases, apresentadas na Figura 3. O processador busca a próxima instrução da memória, guardando-a no registrador de instrução e atualizando o contador de programa. A seguir, ele determina o tipo da instrução e quantos operandos esta instrução utiliza, buscando estes operandos da memória. Finalmente, o processador executa a instrução, armazenando os resultados nos locais apropriados.

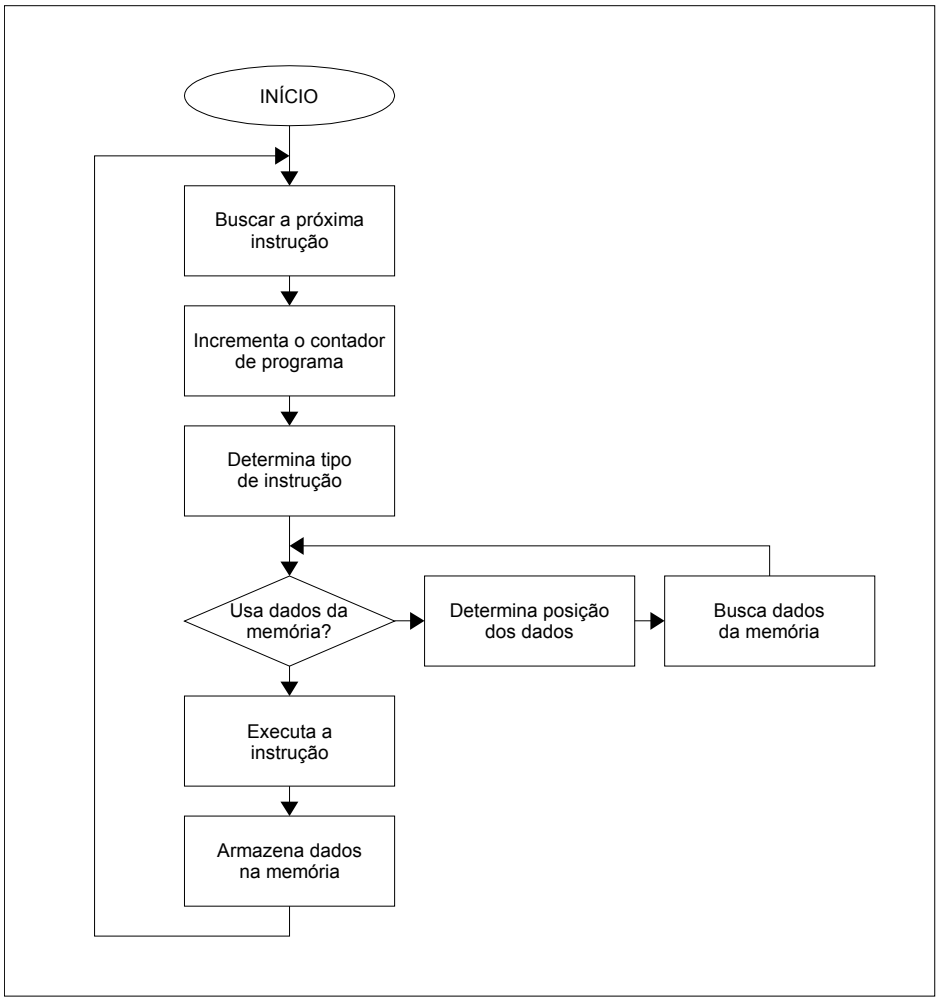


Figura 3 - Ciclo de execução de um microprocessador.

2.1.3 Registradores

O processador conta com uma pequena memória interna, que geralmente não passa de uns poucos bytes (desconsidera-se aqui a memória *cache*, que tem um propósito diferente). Se-

gundo Monteiro (1996, p. 137), "o resultado de uma operação aritmética ou lógica realizada pela UAL[23] deve ser armazenado temporariamente, de modo que possa ser reutilizado mais adiante (por outra instrução) ou apenas para ser, em seguida, transferido para a memória".

Para este propósito, o processador é fabricado com um conjunto de **registradores**. Os registradores são pequenos blocos de memória (geralmente entre 8 e 64 bits). Esta memória é de acesso muito rápido, mais rápido que qualquer outro bloco de memória do computador. Alguns destes registradores ficam disponíveis para serem usados pelo programador (os chamados registradores de propósito geral) enquanto outros têm propósitos específicos (MONTEIRO, 1996).

O **acumulador** é um registrador de propósito geral que é usado especialmente para operações lógicas e aritméticas. Em geral, todas as instruções lógicas e aritméticas de um processador podem ser realizadas utilizando os dados contidos no acumulador. Os modelos mais simples de processador contêm apenas um acumulador, enquanto nos modelos mais complexos encontram-se diversos acumuladores (WEBER, 2000).

O **ponteiro de instruções** (ou IP, do inglês *Instruction Pointer*) é um registrador que armazena o endereço de memória da próxima instrução a ser executada. É também chamado de **contador de programa** (ou PC, do inglês *Program Counter*) (WEBER, 2000). Seu valor geralmente será acrescido do número de bytes da instrução que está sendo executada, exceto nos casos em que a instrução modifica explicitamente o apontador de instruções, como no caso de uma instrução de desvio condicional ou de controle de laço.

O **ponteiro de pilha** (ou SP, do inglês *stack pointer*) é um registrador que armazena o endereço de memória da pilha. A pilha é uma área da memória principal onde o último dado a ser adicionado é o primeiro a ser retirado. Esta área é usada especialmente para armazenar o endereço de retorno em chamadas de sub-rotina, ou para armazenar os valores dos registradores em situações de troca de contexto[24] (MURDOCCA, 2000).

Os **sinalizadores**, ou *flags*, são um conjunto de informações de 1 bit que geralmente são armazenadas em um único registrador. Estas informações são geradas pela ALU, e representam a situação da última operação lógica ou aritmética, freqüentemente através de verdadeiro ou falso. Alguns sinalizadores também são usados para configurar certas funcionalidades do processador. A Tabela 2 exemplifica estes usos, mostrando os sinalizadores usados pelo processador MOS 6502, e são bastante semelhante ao sinalizadores usados por outros processadores. (JACOBS, 2002).

23 Unidade Aritmética e Lógica, o mesmo que ALU.

24 Situação que ocorre em sistemas multitarefa onde, no momento em que o controle é passado de um processo para outro, todas as informações do processo devem ser armazenadas.

Tabela 2 - Sinalizadores do MOS 6502.

Nome	Função
Carry	Verdadeiro se a última operação resultou num transbordamento do primeiro ou do último bit.
Zero	Verdadeiro se o resultado da última operação foi zero.
Interrupt	Enquanto for verdadeiro, o processador não responde a interrupções.
Decimal	Enquanto for verdadeiro, o processador usa o modo decimal nas operações aritméticas.
Break	Verdadeiro se uma interrupção foi gerada na última instrução.
Overflow	Verdadeiro se o resultado da última operação resultou num resultado inválido devido à falta de espaço (por exemplo, quando uma soma de dois números positivos resultou num número negativo devido ao fato do bit mais significativo ter sido mudado de 0 para 1).
Negative	Verdadeiro se a última operação resultou num número negativo.

Fonte: JACOBS, 2002.

2.1.4 Interrupções

Em sistemas multitarefa, é necessário que cada programa receba uma fatia de tempo de processamento, várias vezes por segundo. Isso significa que o processador não pode ficar parado, esperando uma resposta de requisição de E/S[25] (uma requisição para leitura de dados em um disquete, por exemplo) enquanto os outros programas ficam travados. Este problema é resolvido através do uso de **interrupções** (OLIVEIRA, 2001).

Segundo Tanenbaum (1990, p. 42),

> quando a CPU quer realizar E/S, ela carrega um programa especial para um dos canais, e diz ao canal para executá-lo. O canal manipula toda a E/S para e da memória principal, deixando a CPU livre para fazer outras coisas. Quando o canal termina, ele envia à CPU um sinal especial chamado **interrupção**, o que faz a CPU parar o que ela estava fazendo e dar atenção especial ao canal. [grifo do autor]

Uma interrupção é semelhante à chamada de uma sub-rotina; uma rotina é ativada e, no final do seu tratamento, o controle da execução é retornado ao programa principal. Mas como a interrupção pode ser gerada por hardware, ela pode ocorrer a qualquer momento (OLIVEIRA, 2001).

Os processadores geralmente admitem vários tipos de interrupções, cada uma delas identificada por um número. Quando uma interrupção ocorre, o processador consulta a **tabela de vetores de interrupção** na memória principal para descobrir qual o endereço da rotina de tratamento da interrupção equivalente ao periférico que a gerou (OLIVEIRA, 2001).

25 Entrada e Saída, referindo-se à comunicação entre dispositivos, ou à comunicação entre computador e usuário. Também referido pelo termo inglês I/O (*Input/Output*).

Muitas vezes duas ou mais interrupções de hardware podem ocorrer simultaneamente. Para estes casos, existe um controle de prioridades por parte do processador ou de um **periférico controlador de interrupções** (ou PIC, do inglês *Peripheral Interrupt Controller*). Ainda assim, existem interrupções que não podem ser ignoradas – são as **interrupções não-mascaráveis** (ou NMI, do inglês *Non Maskable Interrupt*) (MURDOCCA, 2000).

As interrupções podem ser de três tipos:

- **interrupções de hardware**, geradas por algum componente do computador;
- **interrupções de software** (também chamadas de *traps*), geradas através de uma instrução do próprio processador e geralmente usadas para chamadas de sistema;
- **exceções**, geradas pelo próprio processador em decorrência de um erro, como uma divisão por zero ou a referência a um endereço de memória inexistente (OLIVEIRA, 2001).

2.2 Memória Endereçável

Tanenbaum (1990, p. 30) define a memória como sendo "[...] a parte do computador onde programas e dados são armazenados. [...] Sem uma memória onde os processadores possam ler e escrever informações, não haveria nenhum computador digital de programa armazenado."

Embora haja uma noção popular de que a memória de um computador é uma área contígua que pode ser lida e escrita livremente (como a memória RAM), esta definição está longe de ser verdadeira. Há regiões especiais de memória que mapeiam memória física, memória compartilhada com outros dispositivos (memória de vídeo, por exemplo), registradores de outros dispositivos, ROM[26], espelhamento de memória, etc (DEL BARRIO, 2001).

2.2.1 Memória RAM e ROM

O tipo mais comum de memória endereçável é a **memória RAM**. Esta sigla inglesa traduz-se por Memória de Acesso Aleatório (ou Randômico), ou seja, significa que qualquer região da memória pode ser acessada na mesma quantidade de tempo – ao contrário de uma fita de dados, por exemplo, onde o acesso é seqüencial e leva mais tempo para ler o final da fita do que o seu início. (MURDOCCA, 2000).

Do ponto de vista do nível convencional de máquina, a memória RAM é a mais sim-

26 Memória somente para leitura. Do inglês *Read Only Memory*.

ples de todas – é uma simples memória que permite livre leitura e escrita em qualquer posição. Seus bytes tem, via de regra, 8 bits (exceto em computadores muito antigos).

A **memória ROM** é um tipo de memória que permite somente a leitura. Do ponto de vista do nível convencional de máquina, a memória ROM funciona da mesma forma que a memória RAM, com uma diferença: a memória ROM não pode ser escrita (MURDOCCA, 2000). Exemplos de memória ROM são os cartuchos de videogame ou a BIOS[27] nos computadores.

Existem ainda as memórias do tipo PROM (memórias somente-leitura programáveis, do inglês *Programmable Read-Only Memory*), que são unidades de memória ROM que podem ser reescritas (MURDOCCA, 2000). Do ponto de vista da emulação, este tipo de memória cai em uma das duas categorias acima – geralmente a memória do tipo EPROM[28] (que é reescrita por uma máquina especial) é acessada como uma memória ROM, enquanto a memória do tipo *flash*[29] é acessada de forma parecida com uma memória RAM.

2.2.2 Mapas de Memória

Segundo mencionado anteriormente, a memória endereçável não é um seção contígua que pode ser acessada livremente. Parte desta memória é composta de regiões onde as leituras e escritas serão redirecionadas para um dispositivo, de forma a permitir a comunicação entre o programa e os dispositivos. O mapa contendo estas regiões é chamado **mapa de memória** (ou lista de regiões) (DEL BARRIO, 2001).

Por exemplo, o mapa de memória do videogame portátil *Game Boy*, fabricado pela Nintendo (de 64 kB de memória endereçável, no total) é apresentado na Tabela 3.

Tabela 3 - Mapa de memória do Game Boy

Posição	Tamanho	Nome Interno	Descrição
0000-3FFF	16 kB		Memória ROM (banco 0)
4000-7FFF	16 kB		Memória ROM (banco 1)
8000-9FFF	8 kB	VRAM	Memória de vídeo
A000-BFFF	8 kB		Memória externa
C000-CFFF	4 kB	WRAM	Memória RAM (banco 0)
D000-DFFF	4 kB	WRAM	Memória RAM (banco 1)

27 Sistema Básico de Entradas e Saídas: é o primeiro software que roda em um computador quando este é inicializado, e dá a seqüência para a inicialização do sistema operacional. Em alguns computadores possui também funções de acesso unificado ao hardware. Do inglês *Basic Input/Output System*.

28 Memória Somente Leitura Programável Apagável: tipo de pastilha de memória que pode ser apagado através de raios ultra-violeta e reescrito. Do inglês (*Eraseble Programmable Read-Only Memory*).

29 Memória do tipo PROM de alta velocidade que pode ter bytes específicos reescritos.

Posição	Tamanho	Nome Interno	Descrição
E000-FDFF	7,5 kB	ECHO	Espelho da memória C000-DDFF
FE00-FE9F	160 bytes	OAM	Tabela de atributos de *sprites*
FEA0-FEFF	96 bytes		Não utilizado
FF00-FF7F	128 bytes		Portas de E/S
FF80-FFFE	127 bytes	HRAM	Memória alta
FFFF	1 byte		Registrador de Permissão de Interrupções

Fonte: KORTH, 2001, p. 1

No caso do Game Boy, o programa tem acesso a 64 kB de dados, mas apenas a 8 kB de memória RAM. As duas primeiras áreas de memória (que vão da posição 0x0 à posição 0x7FFF) são a memória ROM – ou seja, quando um cartucho de jogo é inserido no videogame, os dados contidos no cartucho são mapeados para estas posições de memória. Assim, quando o programa estiver acessando uma destas posições, ele estará acessando diretamente os dados do cartucho. Se um cartucho de memória extra for inserido no videogame, ele será mapeado para as posições de 0xA000 a 0xBFFF, e poderá ser usado livremente como a memória RAM (KORTH, 2001).

As posições de 0x8000 a 0x9FFF equivalem à memória de vídeo. Qualquer dado que for escrito em uma destas posições resultará em uma mudança na tela do usuário. Assim, esta região de memória é útil para exibir informações para o usuário, mas não serve para o armazenamento de dados diversos. Estes dados devem ser armazenados na região onde a memória é do tipo RAM.

2.2.3 Espelhamento de Memória

Em alguns casos, partes da memória são espelhadas em outras posições. Isso significa que o acesso a uma posição de memória equivale exatamente ao acesso da posição que a espelha. Por exemplo, em uma máquina hipotética em que a memória de posição 0x0 a 0xFF fosse espelhada nas posição de 0x100 a 0x1FF, um dado escrito na posição 0x10 poderia ser subseqüentemente lido na posição 0x110.

Na realidade, embora exista uma diferença lógica no endereçamento, não há uma diferença física: ambos os endereços apontam para a mesma posição física. Isto é usado especialmente em computadores muito antigos, onde o programa é escrito em linguagem *assembly* levando em conta o menor consumo possível de ciclos de *clock* de um processador. Geralmente, o processador gasta menos ciclos acessando uma memória próxima do que uma memória distan-

te[30] e, portanto, os espelhos de memória são úteis para deixar áreas de memória mais próximas.

Em computadores atuais, no entanto, algumas vezes o espelhamento é feito de forma física. Segundo a Microsoft (2006, p. 1), "memória redundante provê ao sistema um banco de memória de resgate quando um banco de memória falha. O espelhamento de memória divide os bancos de memória em um conjunto espelhado."

Um exemplo de memória espelhada pode ser visto na Tabela 3, onde as posições 0xE000 a 0xFDFF da memória do *Game Boy* espelham as posições 0xC000 a 0xDDFF.

2.2.4 Bancos de Memória

Em computadores mais antigos, o espaço de endereçamento muitas vezes era pequeno demais para acessar toda a memória física de uma vez só. Assim, foram desenvolvidos dispositivos especiais (colocados dentro do microprocessador) que permitissem selecionar qual bloco de memória seria acessado em um determinado momento. Estes blocos de memória física são chamados de **bancos**, e a técnica de troca de bancos é chamada de *bankswitching* (em inglês, literalmente, troca de bancos) (DEL BARRIO, 2001).

Um exemplo disto é o Atari 2600, videogame fabricado pela Atari entre 1977 e 1991. O Atari tem um espaço de apenas 4 kB para endereçamento de ROM, enquanto os jogos produzidos para ele iam de 2 kB a 32 kB. Os jogos de mais de 4 kB usavam bancos de memória, que eram trocados à medida que era necessário. Assim, os jogos de 8 kB usavam 2 bancos, os de 16 kB usavam 3 bancos, e assim por diante (KORTH, 2006).

2.3 Sistema de Vídeo

A visão é o método pelo qual os humanos absorvem a maior quantidade de informação, mais rapidamente. Por esta razão, o vídeo é o método mais importante de saída de informações (DEL BARRIO, 2001).

O sistema de vídeo é composto por duas partes: a **unidade de processamento gráfico** (GPU, do inglês *Graphic Processing Unit*), geralmente composto por uma placa ou um *chip*, e a **saída de vídeo**, que geralmente é um monitor, um *display* de cristal líquido[31] ou uma televisão.

30 Isto não acontece devido ao fato da memória ter diferentes taxas de acesso para diferentes posições, mas sim porque o processador muitas vezes precisa realizar operações como troca de página antes de acessar determinadas partes da memória.

31 Os *displays* são pequenos monitores de cristal líquido usados em dispositivos portáteis.

2.3.1 Saída de Vídeo

A saída de vídeo é responsável por exibir as imagens geradas pela unidade de processamento gráfico (GPU). Geralmente será composta por uma televisão (no caso de computadores pessoais antigos ou videogames), um monitor (no caso de computadores pessoais atuais ou estações de trabalho) ou um *display* de cristal líquido (no caso de dispositivos portáteis ou telefones celulares).

Estes tipos de saída de vídeo usam principalmente três técnicas de exibição de imagens. A primeira técnica desenvolvida, e a que hoje ainda é a mais usada (embora em declínio) é o **tubo de raios catódicos** (CRT, do inglês *Cathode Ray Tube*). Os monitores do tipo CRT são baseados em um feixe de elétrons que desenha, linha por linha, a imagem em uma tela de fósforo. Quando o fósforo é atingido pelo feixe de elétrons, ele brilha a uma determinada intensidade, compondo a imagem na tela. Como o fósforo mantém a cor apenas por um curto espaço de tempo, a imagem precisa ser redesenhada várias vezes por segundo (MONTEIRO, 1996). Veja a Figura 4.

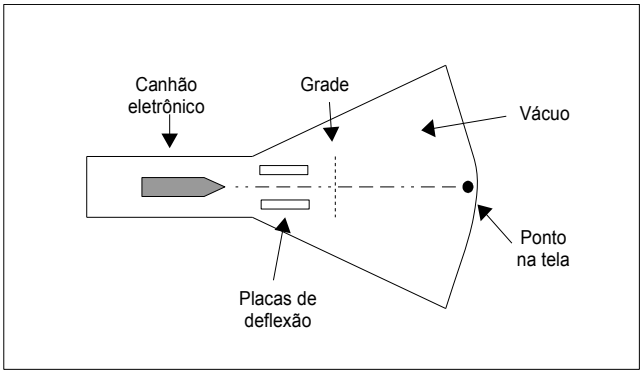


Figura 4 - Seção de um CRT. Fonte: TANENBAUM, 1990

Os outros técnicas comuns para a exibição de imagens são o **cristal líquido** e o **plasma**. O monitor de cristal líquido utiliza moléculas de cristal líquido para exibição de imagens, enquanto o monitor de plasma utiliza uma mistura química de xenon e neon. No entanto, ambos os tipos de monitor têm o funcionamento semelhante a um monitor CRT quando conectados a um computador e, por isso, este será o tipo de monitor que este trabalho focará com mais profundidade (MONTEIRO, 1996).

2.3.1.1 O Caminho do Feixe de Elétrons

Os primeiros computadores utilizavam um sistema vetorial, onde o feixe de elétrons era movimentado livremente. Por exemplo, se o programa quisesse desenhar um triângulo na tela, o feixe era movimentado desenhando rapidamente três linhas, equivalentes aos três lados do triângulo.

Este método logo caiu em desuso devido a duas limitações. A primeira era que apenas os formatos das imagens podiam ser desenhados, mas elas não podiam ser preenchidas. Outro problema era que as imagens tinham que ser redesenhadas continuamente e, se houvessem muitas imagens na tela, havia uma grande sensação de *flicker*[32] (WIKIPEDIA, 2006b).

Por estas razões, os monitores vetoriais foram substituídos pelos monitores do tipo *raster* (do inglês "rastro"). Em um monitor do tipo *raster* o feixe de elétrons desenha toda a tela seqüencialmente, de cima para baixo e da esquerda para a direita, várias vezes por segundo. Desta forma, a tela tem um número determinado de "pontos", que são o menor elemento de imagem possível de ser mostrado em uma só cor. Estes pontos são chamados *pixeis*[33] (DEL BARRIO, 2001).

O caminho percorrido pelo feixe de elétrons pode ser visto na Figura 5. A compreensão deste conceito é fundamental para a construção de emuladores.

A primeira operação a ser executada é o **retraço vertical** (ou VSYNC, do inglês, *vertical synchronization*, ou sincronização vertical). No retraço vertical, o monitor recebe a informação de que o feixe de elétrons deve se deslocar para o canto superior esquerdo do usuário. Durante este tempo (que num televisor leva o equivalente à varredura de 3 linhas) o sinal é desligado, caso contrário a imagem na tela ficaria deformada (MONTEIRO, 1996).

32 Sensação de que a imagem está "piscando" muito rapidamente na tela. Isto ocorre devido ao redesenho muito lento da imagem, e pode causar desconforto e dores de cabeça no usuário.

33 Do inglês *picture element*, é a menor quantidade de informação possível de ser exibida em um monitor, ou seja, cada um dos pontos da imagem. Este elemento de informação não é um ponto ou um quadrado, mas sim uma amostra abstrata.

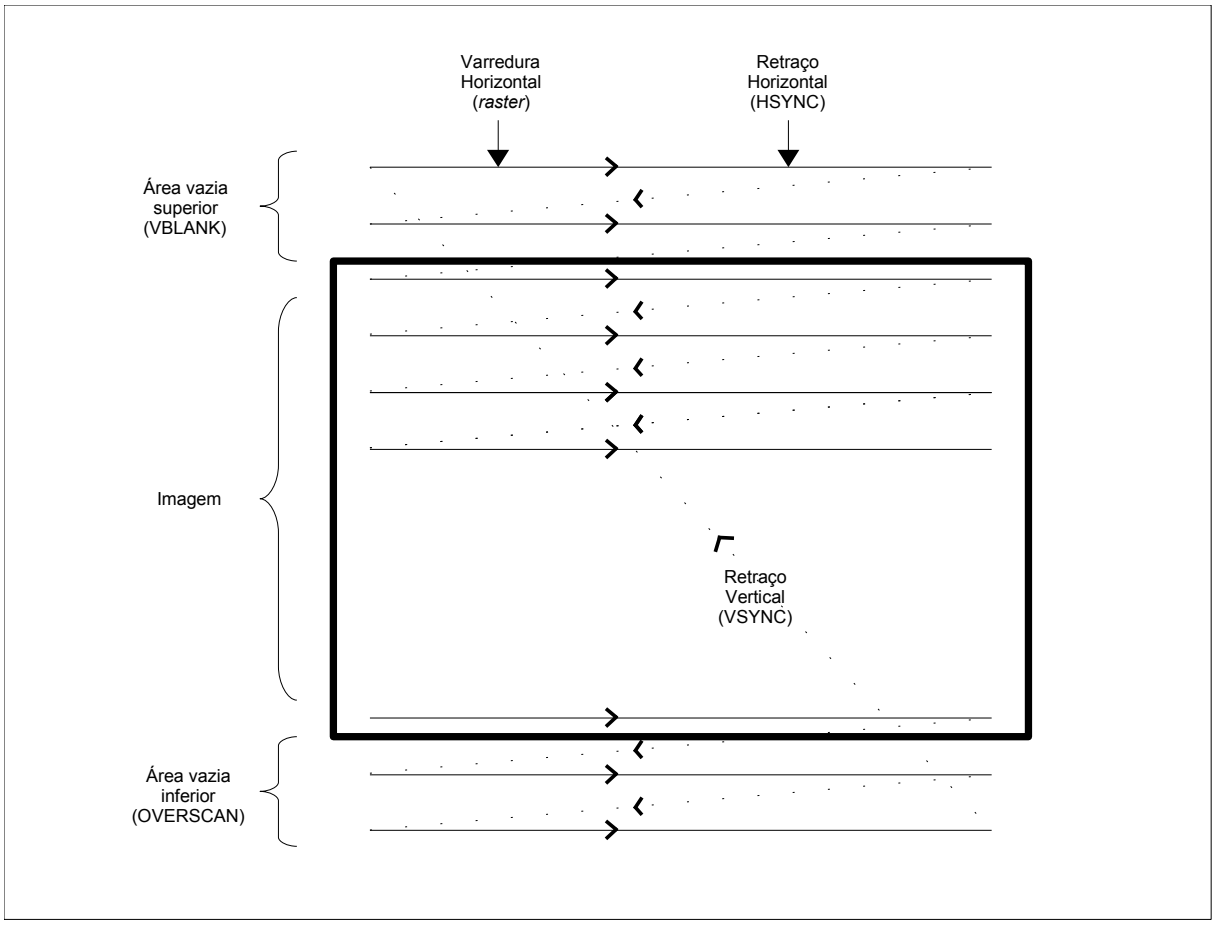


Figura 5 - Caminho percorrido pelo feixe de elétrons em um monitor CRT.

Existem vários modelos de televisores e monitores, alguns exibindo um espaço de imagem extra na parte superior, outros na parte inferior, e outros em nenhuma. Assim, os computadores e videogames que se conectam em televisores geralmente usam um padrão onde a parte superior (VBLANK, do inglês *Vertical Blank*, ou espaço vertical em branco) e a parte inferior da imagem (OVERSCAN, do inglês, varredura posterior) não são exibidas. Segundo pesquisas realizadas pela empresa norte-americana Atari, um computador usando um VBLANK de 37 linhas de varredura e um OVERSCAN de 30 linhas de varredura funcionará em todos os tipos de televisores (WRIGHT, 1979). Monitores modernos não têm as áreas de VBLANK e OVERSCAN.

Após o VBLANK (ou o VSYNC em monitores modernos), a imagem começa a ser desenhada, linha por linha. Logo após a primeira linha ser desenhada (da esquerda para a direita), o elétron precisa ser trazido de volta para o lado esquerdo da tela. O período em que isto acontece é chamado **retraço horizontal** (HSYNC , do inglês *Horizontal Synchronization*, ou sincronização horizontal) (MONTEIRO, 1996). O retraço horizontal leva, em média, um quarto do tempo necessário para a varredura, e durante este tempo a geração de imagens é, obviamente, desligada (WRIGHT, 1979).

Depois da imagem ser desenhada até embaixo, o feixe de elétrons passa a área de OVERSCAN e novamente acontece o retraço vertical, assim sucessivamente, várias vezes por segundo.

O freqüência com que a imagem é desenhada na tela é chamada de **freqüência vertical** ou taxa de atualização. Esta taxa é medida em Hertz, e a medida é exatamente o número de vezes que a imagem é atualizada por segundo (por exemplo, um monitor operando a 60 Hz atualiza a imagem 60 vezes por segundo) (MONTEIRO, 1996).

Os televisores usam uma taxa fixa de atualização, que depende do país de origem. As duas principais taxas são as de 60 Hz (usada na América do Norte) e de 50 Hz (usada no Brasil e na Europa) (MURDOCCA, 2000).

Este fator é importante e deve ser levado em consideração na construção de emuladores de computadores e videogames mais antigos. No caso do Atari 2600, por exemplo: se um jogo escrito para um sistema de 50 Hz for executado num sistema de 60 Hz, o jogo rodará 17% mais rápido que no sistema original (porque, neste tipo de arquitetura, a velocidade do retraço vertical controla a velocidade de todo o resto do hardware) (WRIGHT, 1979).

Os monitores mais modernos aceitam várias freqüência verticais, geralmente variando entre 50 Hz e 90 Hz. Como os monitores modernos normalmente não utilizam entrelaçamento[34], a freqüência precisa ser maior que a da televisão para não irritar o olho humano.

2.3.2 Unidade de Processamento Gráfico

Uma unidade de processamento gráfico (GPU) é um *chip* ou uma placa presente em um computador que gera as imagens que serão enviadas para o monitor. As GPU também são conhecidas como placas de vídeo, VPU (*Video Processing Unit*, do inglês Unidade de Processamento de Vídeo) ou VDP (*Video Display Processor*, do inglês Processador de Exibição de Imagens).

Existem vários sistemas de GPU, das quais este trabalho descreverá os principais. É importante lembrar que uma determinada placa pode ter mais de um sistema, embora use apenas um por vez. Exemplo disto são as placas atuais de PC, onde elas podem usar o modo terminal (modo texto), *framebuffer*[35] (modo gráfico 2D) ou o modo gráfico tridimensional.

---

34 O entrelaçamento é uma técnica onde apenas a metade das linhas é atualizada em cada retraço vertical, na primeira vez as linhas pares, e na segunda as ímpares, sucessivamente. Isto faz com que um televisor operando a 50 Hz só complete um retraço completo 25 vezes por segundo, sem que o olho humano consiga perceber.

35 Sistema onde a memória possui um mapa de bits exatamente igual às informações que estão sendo exibidas na tela.

2.3.2.1 Gráficos Vetoriais

Este modo já foi descrito no início seção 2.3.1.1. Ao invés do tipo *raster*, onde os gráficos são desenhados na tela de cima a baixo, este modo controla diretamente o feixe de elétrons, usando-o para desenhar as formas na tela. É bastante usado em osciloscópios, mas muito raramente em computadores ou videogames. Uma exceção disto é o videogame Vectrex, produzido pela companhia americana General Consumer Electric entre 1982 e 1984 (WIKIPEDIA, 2006c).

Uma figura do videogame Vectrex pode ser visto na Figura 6. Esta imagem foi gerada a partir do emulador Vecx. É possível notar a falta de cores e de preenchimento nas figuras. Como este tipo de gráfico foi raramente utilizado, a emulação dele não será mencionada neste texto. Todos os outros tipos de gráficos usam o modo *raster* de retraço no monitor.

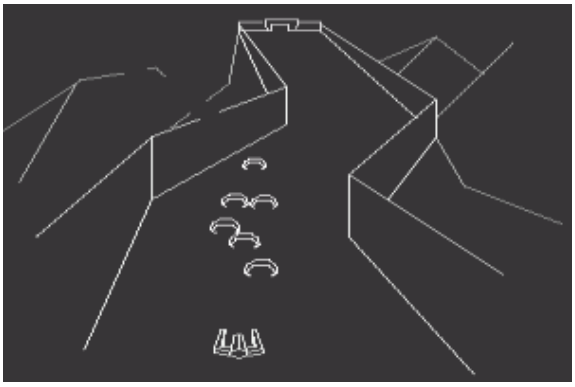


Figura 6 - Imagem do videogame Vectrex.

2.3.2.2 Gráficos Sincronizados

As GPU de gráfico sincronizados são um tipo de placas gráficas que possuem apenas uma pequena quantidade de memória. Como a memória é pequena, não é possível armazenar as posições dos pixeis, e o programa que está sendo executado precisa estar constantemente atualizando a tela de forma manual.

Um exemplo deste tipo de placa gráfica é o TIA (Adaptador de Interface de Televisão, do inglês *Television Interface Adapter*), que era usado no videogame Atari 2600. O atualização de tela do Atari 2600 tinha quatro fases: (i) 3 linhas de VSYNC; (ii) 37 linhas de VBLANK; (iii) 192 linhas de imagem; e (iv) 30 linhas de OVERSCAN. Assim, um jogo precisava estar sincronizado com o feixe de elétrons da televisão. Durante as 70 linhas de VSYNC, VBLANK e OVERSCAN, ele podia fazer o processamento do jogo (contar os pontos, verificar colisões, verificar dispositivos de entrada, etc), mas durante as 192 linhas de imagem, o processamento pre-

cisava parar para que o programa fizesse o desenho na tela. Isto acontecia 60 vezes por segundo (ou 50, em televisores do tipo PAL). Isso significava que um jogo de Atari 2600 gastava cerca de 75% do seu tempo gerando a imagem, e apenas 25% fazendo o processamento (ISRAEL, 2004).

Por isto, a programação de jogos de Atari 2600 era considerada uma tarefa desafiadora, pois o programador precisava constantemente manter em mente o número de ciclos de *clock* de cada uma de suas instruções. Se o processamento acabasse ultrapassando as 70 linhas disponíveis para isso (ou, por descuido, o programador fizesse uma rotina que usasse menos de 70), a imagem ficaria fora de sincronia, resultando num programa inutilizável.

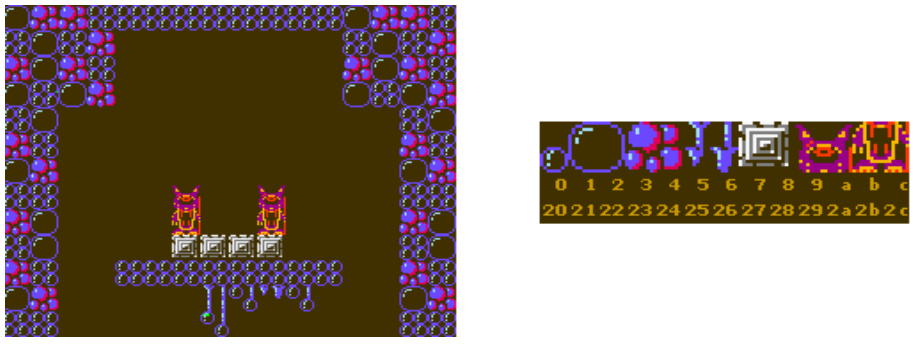
Vários programas utilizavam técnicas pouco ortodoxas de programação, de modo a obter o máximo do processador gráfico. Um exemplo disto é o Video Chess, um jogo de xadrez desenvolvido pela Atari em 1978 para o Atari 2600. O jogo permitia que o usuário jogasse xadrez contra o computador. Durante o processamento da jogada do computador, a tela ficava preta. Assim, o jogo não gastava ciclos com a atualização da imagem, de modo a acelerar o processamento. Uma vez que o computador estivesse terminado de processar a sua jogada, a imagem voltava a ser exibida.

2.3.2.3 *Tiles* e *Sprites*

Segundo del Barrio (2001, p. 110), "motores gráficos baseados em *tiles* e *sprites* eram o tipo mais comum em videogames nos anos 80 e 90. Eles eram muito populares devido à sua habilidade em produzir bons gráficos e animações sem muito uso de CPU ou memória."

Nos jogos de computador e videogame, é muito comum existir uma repetição de padrões de imagem, representando terreno, cenário, parede de tijolos, etc. Um exemplo de uma imagem com estes padrões repetidos pode ser visto na Figura 7.

Figura 7: Exemplo de mapa de tiles.



Fonte: CLD, 2006

Observando-se a imagem da esquerda na Figura 7, nota-se a existência de padrões repetidos – as paredes, pedras, etc. Na imagem da direita, está um mapa com a legenda destes padrões. Estes padrões repetitivos são chamados de *tiles* (do inglês "azulejos").

O uso de *tiles* minimiza bastante o uso de memória. Em uma posição da memória é guardado um conjunto de *tiles* (chamado de *tileset*, do inglês "conjunto de *tiles*"), como o da imagem da esquerda na Figura 7, enquanto em outra posição da memória é guardado um mapa das posições dos *tiles* na tela (chamado *tilemap*, do inglês "mapa de *tiles*"). Desta forma, é possível repetir um *tile* quantas vezes forem necessárias, sem ter que copiar toda a memória várias vezes. Isto, além de minimizar o uso de memória, acelera o processamento, permitindo que a cópia de um *tile* seja realizada apenas com uma ou duas instruções do processador, bem facilitando como a rolagem de tela.

Os *sprites* funcionam de maneira semelhante aos *tiles*, porém são mais completos. Enquanto os *tiles* geralmente são usados para o fundo da imagem, os *sprites* são usados para as personagens. Assim, o computador geralmente permite um ajuste mais exato da posição do *sprite*, permite que hajam modificações no formato do *sprite* e, como o *sprite* normalmente vai aparecer à frente dos *tiles*, parte dele é transparente.

Existe um tratamento de prioridades neste tipo de processador gráfico. Na maioria dos casos há mais de uma camada de *tiles* e mais de uma camada de *sprites*. Cada uma das camadas tem uma prioridade diferente – assim, quando dois *tiles* ou *sprites* ocuparem a mesma posição na tela, o que tiver a prioridade menor ficará escondido debaixo daquele que tem a prioridade maior. Embora os *sprites* geralmente tenham uma prioridade maior que os *tiles*, este nem sempre é o caso. Alguns jogos, por exemplo, podem desejar que a personagem apareça à frente do cenário mas passe por detrás de uma coluna – neste caso, os *tiles* da coluna terão prioridade superior ao *sprite* do jogador.

Este tipo de processador gráfico geralmente possui tratamento de colisões. Assim, o jogo pode requisitar à placa de vídeo a informação sobre se dois *sprites* estão se sobrepondo (colidindo) na tela em um determinado momento. Num jogo de naves, por exemplo, é possível saber se o *sprite* do míssil disparado pela nave do jogador está colidindo com a nave adversária, acionando assim a subrotina que destrói a nave em questão (DEL BARRIO, 2001).

2.3.2.4 Framebuffer

O sistema gráfico mais simples de ser emulado é o *framebuffer*. Neste sistema existe na memória um mapa exato dos *pixeis* que aparecem na tela. Assim, se o programa escreve um

byte na área da memória de vídeo, na próxima varredura vertical o pixel aparecerá automaticamente na posição equivalente. A palavra *framebuffer* significa "memória interna de quadro", já que o quadro inteiro da imagem é guardado na memória.

O *framebuffer* é, portanto, uma matriz de dados do tamanho da tela. Se a tela tiver 320x200 pixels, a matriz do *framebuffer* será de 640 por 480 (ou seja, uma matriz de 307.200 dados). O tamanho do dado vai depender do sistema usado (com ou sem paleta) e do número de cores desejadas.

No sistema sem paleta, cada dado da matriz contém a informação de cor usando algum sistema de armazenamento de cores, como o RGB[36]. Quanto maior o dado, maior o número de cores que podem ser exibidas. Se o dado for de 8 bits, 256 cores podem ser exibidas. Nos computadores atuais, é comum o uso de 32 bits, permitindo a exibição de 4 bilhões de cores simultâneas.

No sistema com paleta, existe um mapa de cores, com um número correspondente a cada cor. No dado da matriz, é armazenado o número da cor. Desta forma é possível exibir, por exemplo, um universo de 4 bilhões de cores usando apenas 1 byte por pixel. Mas neste caso, apenas 256 cores podem ser exibidas simultaneamente.

O sistema *framebuffer* utiliza muito mais memória que os outros sistemas. Um exemplo seria a definição de vídeo usada amplamente nos Pcs atuais: 1024x768 pixeis, com 32 bits de cores. Isto significa que a memória de vídeo necessita de, no mínimo, 3,14 Mb (1024 x 768 x 4 bytes).

Para as operações gráficas, é usada uma técnica chamada *blit* (do inglês BLT - *Bit Block Transfer*, ou "transferência de bloco de bits"). Neste caso, uma grande quantidade de bits é transferida de uma porção da memória para outra, gerando assim a imagem. Por exemplo, se uma figura fosse ser exibida na tela, a operação de *blit* copiaria todos os bytes da memória principal (onde a figura está armazenada) para a memória de vídeo, exibindo assim a imagem para o usuário. A técnica de *blit* geralmente faz algum tipo de modificação ou conversão na imagem, mudando sua resolução e tornando transparente as áreas que não devem ser exibidas.

A maior parte dos *framebuffers* atuais tem mais memória disponível do que o necessário. Isto permite o uso de uma técnica para acelerar a exibição gráfica, que é o armazenamento das imagens a serem coladas na tela principal na própria memória gráfica. Como a maioria das placas modernas possuem a técnica de *blit* implementada em hardware, a operação é muito mais

36 Sistema de armazenamento de cor onde a cor é formada pela mistura das intensidades de vermelho, verde e azul (por exemplo, a cor 0x2040A0 teria 0x20 de vermelho, 0x40 de verde e 0xA0 de azul, sendo 0x0 a ausência de cor e 0xFF a saturação total). Do inglês *Red Green Blue*, ou Verde Vermelho Azul.

rápida. Este tipo de placa é geralmente conhecida por “placa acelerada 2D” (DEL BARRIO, 2001).

Em algumas situações, a memória extra é igual ou maior que o tamanho de um quadro inteiro. Nestas ocasiões, pode-se usar uma técnica chamada de *page flipping*, onde enquanto o quadro de exibição está sendo mostrado na tela, o quadro seguinte já pode ir sendo montado. Quando é feito o retraço vertical, o novo quadro é exibido e o que acabou de ser exibido se torna o secundário, através de uma instrução executada pelo acelerador gráfico (WIKIPEDIA, 2006d).

2.3.2.5 Terminais de Texto

O terminal de texto é um sistema gráfico que permite apenas a exibição de texto. São usados em praticamente todos os computadores, e funcionam internamente de maneira parecida com o sistema de *frambebuffer*. A diferença é que a matriz, ao invés de guardar informações de pixeis, guarda informações de caracter. Esta informação geralmente é guardada em 2 bytes: um contendo o código do caracter, e outro contendo as informações de atributo (como cor da letra e cor do fundo).

O sistema IBM PC usa, por padrão, 4000 bytes de memória de vídeo para este modo: são 25 linhas e 80 colunas de texto, cada caracter usando 2 bytes. Este sistema é, portanto, muito mais econômico que qualquer outro (TANENBAUM, 1990).

2.3.2.6 Placas Gráficas Tridimensionais

Placas gráficas tridimensionais ou placas 3D são processadores gráficos que contém um número de instruções criadas especificamente para exibição de gráficos tridimensionais, como tratamento de volume, distância, sombreamento, etc. Estas instruções variam de placa para placa, mas geralmente o acesso a elas é feita de forma unificada através de uma biblioteca como o OpenGL, Mesa 3D ou Direct3D.

Devido à complexidade deste tipo de hardware, este assunto não será abordado neste trabalho.

2.4 Sistemas de Entrada

Dispositivos de entrada são dispositivos usados por um usuário para inserir dados em um computador. Dispositivos de entrada comuns em computadores são teclados e mouses, e joysticks nos videogames.

Toda vez que o usuário executa uma operação em algum destes dispositivos (por exemplo, pressiona uma tecla no teclado), esta informação é armazenada em algum lugar da memória. Existem duas técnicas para detectar quando o usuário executou alguma operação (e, portanto, um dado da memória foi modificado): *polling* e interrupção.

A técnica de ***polling*** (do inglês "sondagem") é a mais simples das duas e geralmente é usada para sistemas monotarefa[37]. Neste tipo de sistema, o programa precisa verificar freqüentemente se alguma das posições de memória referentes ao dispositivo de entrada foi modificada, e agir de acordo.

Como isto se torna complicado nos sistemas multitarefa, existe outro sistema que onde cada dispositivo tem um **controlador**, que gera uma **interrupção** de hardware para o processador toda vez que o usuário efetua alguma operação no dispositivo (MURDOCCA, 2000).

2.5 Considerações Finais

Uma arquitetura de hardware é formada por um grande número de componentes, e a natureza expansível dos computadores modernos faz com que o número de componentes conectáveis seja ainda maior. Este trabalho se deterá apenas nos principais componentes, presentes em todos os computadores: microprocessador, memória, vídeo e sistemas de entrada. Dos sistemas de vídeo mencionados, serão tratados apenas os síncronos, *tiles/sprites, framebuffer* e terminais de texto.

37 Sistema que, ao contrário dos sistemas multitarefa, executa apenas um programa por vez. Exemplos são o sistema operacional DOS e os videogames.

3 TÉCNICAS TRADICIONAIS NA CONSTRUÇÃO DE EMULADORES

Segundo foi explicado no Capítulo 1, um emulador é um programa que imita o comportamento de uma arquitetura de hardware, permitindo que programas escritos para aquela arquitetura possam ser executados em outra. Este capítulo discute a forma como tradicionalmente são construídos os emuladores.

3.1 Estrutura Básica de um Emulador

Existem algumas diferenças fundamentais entre a construção de uma arquitetura física de hardware e a construção de um emulador.

A primeira diferença fundamental é que os diferentes componentes de hardware funcionam em paralelo, enquanto eles precisam ser emulados um após o outro em um emulador. Isto acontece devido ao fato de que um programa, a não ser que esteja sendo executado em um computador multiprocessado (isto é, que tenha mais de um processador rodando em paralelo), pode apenas executar uma operação de cada vez. Os sistemas operacionais multitarefa dão ao usuário a impressão de que os programas estão rodando em paralelo ao cada um em uma fatia de tempo muito pequena (geralmente medida em milésimos de segundo) repetidamente (DEL BARRIO, 2001).

E é exatamente este método que é usado para fazer com que o usuário tenha a ilusão de que os vários componentes do emulador estão rodando em paralelo, que é fazer a emulação de um dispositivo um após o outro, milhares de vezes por segundo. Isto se atinge através de um laço, que para um emulador simples poderia ser o seguinte:

```
1       int ciclos;
2
3       while(continuar_emulacao)
4       {
5           emular_cpu(&ciclos);
6           emular_graficos(ciclos);
7           emular_som(ciclos);
8           emular_outros_dispositivos(ciclos);
9       }
```

No código acima, é primeiro feita a emulação do microprocessador, depois dos gráficos, do som e assim por diante. Como a execução este laço acontece milhares de vezes por segundo, o usuário não percebe que os dispositivos não estão sendo emulados simultaneamente.

Uma segunda diferença básica acontece em relação ao controle do tempo. Em uma arquitetura física de hardware, o sinal de *clock* é gerado por um cristal com uma certa freqüência, de modo a manter todos os dispositivos em sincronia. Já em um emulador, isto não acontece desta forma. Neste caso, o primeiro dispositivo a ser emulado, que é o microprocessador "gastar" um determinado número de ciclos de *clock* a cada operação (interpretação de uma instrução). Esta informação é então passada a cada um dos dispositivos que serão emulados a seguir, de forma que eles executem o número de instruções equivalente àquele número de ciclos de *clock* (DEL BARRIO, 2001).

Por exemplo, imagine-se a situação hipotética onde o emulador vai iniciar a emulação de um bloco de código. A função que emula o processador lê o primeiro byte da memória e descobre que este byte equivale a uma instrução de incremento de um registrador. A função executa então o incremento, retornando ao laço principal a informação de que a instrução consumiu 5 ciclos de *clock*. O emulador passa então à função que emula o vídeo, informando que esta deve executar o equivalente a 5 ciclos de *clock* (por exemplo, desenhar 5 pixeis na tela). Após isto o emulador passa à função que emula o som, emulando 5 ciclos de *clock*, e assim por diante, até que o laço esteja completo e o processo se inicie novamente com a próxima instrução.

No código mostrado acima, isto pode ser visto através da variável ciclos. Na primeira chamada à função emular_cpu, a variável é passada por referência[38], de modo que o número de ciclos gastos possa ser retornado. A seguir, a variável ciclos é passada por valor[39] para as outras funções, para que estas tenham condições de saber o quanto devem emular antes de retornar o

38 Quando uma variável é passada para uma função por referência, isto significa que, na verdade, um ponteiro da variável é passado. Isto permite que o valor da variável seja modificado dentro da função, e esta modificação se reflita fora da função.

39 Quando uma variável é passada para uma função por valor, isto significa que, na verdade, uma cópia da variável é passada. Qualquer modificação feita à variável se refletirá apenas dentro do escopo da função, sendo que uma vez que o controle seja retornado pela função, o valor da variável será o mesmo que antes da chamada.

controle ao laço principal.

Em muitos casos, dois componentes não funcionam na mesma velocidade, mas ao invés disto usam um **multiplicador de *clock***. Neste caso é necessário que se façam as devidas conversões. Um exemplo disto é o caso do Atari 2600. Nesta arquitetura, o processador roda a 1,19 Mhz, enquanto a placa de vídeo roda a 3,58 Mhz (portanto, 3 vezes mais rápido) (WRIGHT, 1979). Isto significa que o número de ciclos deve ser multiplicado por 3 quando for passado para a emulação do vídeo; por exemplo, se uma instrução executada pela CPU levar 5 ciclos, um total de 15 ciclos deve ser passado ao vídeo.

3.2 Emulação do Microprocessador

A emulação do microprocessador é crucial para o desempenho do emulador, especialmente nos computadores modernos onde a CPU tem uma velocidade bastante alta. Na seção 1.5 discutiram-se as diferentes técnicas utilizadas na construção de um emulador. Todas estas técnicas dizem respeito especialmente à emulação do microprocessador, e este trabalho se deterá na emulação por interpretação.

3.2.1 Registradores

É necessário manter as informações dos registradores e das *flags* entre a execução de uma instrução e outra. Estas informações são guardadas em variáveis.

Como a quantidade de memória gasta com os registradores é pequena, e a velocidade de acesso a eles é crucial, deve-se investir no acesso rápido, nem que haja um gasto de memória maior que o necessário. Geralmente vale a pena dividir o registrador de *flags* em várias variáveis, umas para cada *flag*. Isto torna o acesso mais rápido. Fayzullin (2006, p. 7) acrescenta ainda:

> Tente usar apenas inteiros do tamanho da base suportada pelo microprocessador, isto é, use *int* ao invés de *short* ou *long*[40]. Isto reduzirá a quantidade de código que o compilador gera para fazer a conversão entre diferentes comprimentos de inteiros. Também diminuirá o tempo de acesso à memória, pois algumas CPUs funcionam mais rapidamente quando estão lendo ou escrevendo dados do tamanho da base alinhada ao tamanho da base do limite do endereço. [tradução nossa]

Alguns registradores permitem um uso da palavra reservada *register*, que faz com que aquela variável seja armazenada em um registrador, de modo a acelerar o acesso. Isto é uma boa idéia para o armazenamento de registradores de acesso mais comum, como o ponteiro de instruções e o acumulador. No entanto, a maioria dos compiladores modernos já calcula qual a variá-

40 *int*, *short* e *long* são três tamanhos de variáveis inteiras usadas pela linguagem de programação C, sendo respectivamente os tamanhos médio, curto e longo.

vel será mais acessada e a coloca no registrador automaticamente.

3.2.2 Seqüência de Operação

A seção 2.1.2 apresentou o ciclo **busca-decodificação-execução** usado pelo processador para seu funcionamento, e é baseado neste ciclo que a emulação de um processador funciona.

O primeiro passo é buscar na memória o byte para o qual o contador de instruções aponta. Baseado neste byte, o processador pode saber se ele precisa buscar mais bytes da memória para complementar a instrução, isto é, se a instrução possui parâmetros. Isto acontecerá apenas em processadores CISC. Nos processadores RISC, todas as instruções acrescidas de seus parâmetros possuem o mesmo comprimento, e este passo separará a instrução de seu parâmetro.

O segundo passo é a decodificação, ou seja, descobrir qual instrução equivale àquele byte buscado na memória. Existem várias maneiras de fazer isto, e a maneira mais comum é através de uma tabela de saltos. Isto é exemplificado no seguinte código:

```
1     void emular_processador(int &ciclos)
2     {
3         unsigned int instrucao = busca_instrucao();
4         unsigned int parametro;
5
6         switch(instrucao)
7         {
8             case 0x01:
9                 ciclos = 3;
10                /* emular instrução 0x01 */
11                break;
12            case 0x02:
13                parametro = busca_parametro();
14                ciclos = 7;
15                /* emular instrução 0x01 */
16                break;
17            ...
18            case 0x..N:
19                /* emular instrução 0x..N */
20                break;
21            default:
22                instrucao_ilegal();
23        }
24    }
```

Neste caso, o próximo byte de memória é buscado na linha 3. Na linha 3, a instrução *switch* determina que o fluxo do código deve saltar para a instrução que foi buscada da memória. Em cada instrução, a variável ciclos é alimentada, de modo que os outros dispositivos sai-

bam quantos ciclos foram gastos com a instrução. Caso a instrução não esteja na lista, é uma instrução ilegal e deve ser tratada de acordo (com a reinicialização do computador, por exemplo, ou uma mensagem ao usuário).

Na linha 13 é ainda apresentado o exemplo de uma instrução que usa um parâmetro. Neste caso, o parâmetros deve ser buscado da memória e o apontador de instruções avançado em mais um byte.

Existem ainda outras formas de implementação. Uma delas é através de blocos se-então, como no exemplo abaixo:

```
1     unsigned int instrucao = busca_instrucao();
2     if (instrucao == 0x01)
3     {
4         /* executa instrução 0x01 */
5     }
6     else if (instrucao == 0x02)
7     {
8         /* executa instrução 0x02 */
9     }
10    ...
11    else if (instrucao == 0x0..N)
12    {
13        /* executa instrução 0x0..N */
14    }
15    else
16        instrucao_ilegal();
```

O tipo de implementação a ser adotado vai depender do compilador usado. Alguns compiladores farão com que o programa produzido pelo primeiro código seja mais lento, devido ao grande número de saltos realizados, enquanto outros compiladores farão com que o segundo código seja mais lento, devido ao número de comparações envolvidas.

Outra opção é o uso de um *array*[41] de funções para a execução das instruções. O grande problema deste método é que o desempenho pode ser drasticamente reduzido devido à geração de *overhead* (ver seção 3.5.1).

Na maioria dos casos, as CPUs terão mais de uma instrução para executar a mesma operação. Por exemplo, uma determinada CPU pode ter uma instrução de subtração entre dois registradores, entre um registrador e uma posição de memória, e entre um registrador e um valor passado por parâmetro. Neste caso, se torna útil o uso de macros[42].

Levando em consideração o exemplo acima, o código deste emulador poderia ser o se-

41 Estrutura de dados contendo elementos homogêneos de um tipo de dados específico. Do inglês “vetor”.
42 Macros são identificadores que são substituído por blocos de código previamente especificados.

guinte:

```
1     unsigned int A;  /* acumulador */
2     unsigned int X;  /* registrador geral */
3     unsigned int Z;  /* flag de zeramento */
4
5     #define SUBTRAI(x,y) \
6         if((x - y) > 0)  \
7             Z = 1;       \
8         else             \
9             Z = 0;       \
10        A = (x - y);
11
12    void emula_cpu()
13    {
14        unsigned int parametro;
15        unsigned int instrução = busca_instrucao();
16
17        switch(instrucao)
18        {
19            ...
20            case 0xA0: /* SUB A,X */
21                SUBTRAI(A, X);
22                break;
23            case 0xA1: /* SUB A,mem */
24                parametro = busca_parametro();
25                SUBTRAI(A, ram[parametro]);
26                break;
27            case 0xA2: /* SUB A,val */
28                parametro = busca_parametro();
29                SUBTRAI(A, parametro);
30                break;
31            ...
32        }
33    }
```

No caso apresentado, as linhas de 5 a 10 contém uma macro que subtrai dois valores armazenando o resultado em um acumulador, e colocando na *flag* de zeramento se o resultado foi zero ou não. Na tabela de saltos aparecem as três formas de chamada da função de subtração, conforme exemplificado acima. Nestes casos, é melhor usar uma função do que uma macro, porque com uma macro não há formação de *overhead* (ver seção 3.5.1).

3.2.3 Emulação das Instruções

3.2.4 Interrupções

3.3 Memória

3.4 Gráficos

3.5 Velocidade da Emulação

Uma das coisas mais importante que deve ser levada em consideração na construção de um emulador é o *feeling*, ou seja, a sensação do usuário de estar executando a máquina real. Para isto, é necessário que a velocidade de execução do emulador seja o mais próximo possível do hardware original (nem mais lento, nem mais rápido). Além disso, o emulador deve rodar de forma suave, isto é, não deve haver momentos onde a execução fica mais rápida ou mais lenta (ao menos não de forma perceptível ao usuário) (DEL BARRIO, 2001).

Manter a mesma velocidade do hardware original é tarefa desafiadora. Um emulador precisa executar milhares de instruções por segundo, e por isso o código precisa ser o mais otimizado possível. O uso de um *profiler*[43] pode auxiliar na tarefa de encontrar quais linhas de código são executadas com maior freqüência, permitindo ao programador otimizá-las (FAYZULLIN, 1997).

Há uma grande diferença de performance entre a execução de programas escritos em diferentes linguagens de programação. É claro que a velocidade de execução vai depender do quão bem escrito é o código mas, em geral, programas escritos em assembly costumam ser mais rápidos que os demais, devido à proximidade da máquina (uma vez que estão se emulando processadores, quanto mais controle do processador melhor). A grande desvantagem do assembly é a falta de portabilidade, uma vez que um programa escrito em assembly não poderá ser executado em outro processador (e dificilmente em outro sistema operacional que use o mesmo processador).

A maioria dos emuladores utiliza assembly ou C[44]. O uso de C é considerado a melhor opção para a escrita de emuladores portáveis, devido à sua proximidade ao hardware e capacidade de manipulação direta de memória. Alguns emuladores utilizam o C++[45], fazendo uso das

43 Programa que analisa a performance de outro programa, identificando quais são as funções mais executadas, e o tempo de execução de cada uma.

44 Linguagem de programação imperativa e funcional criada por Dennis Ritchie e Ken Thompson nos anos 70. É uma linguagem de alto nível, mas que permite o acesso direto à memória e a partes do hardware. Linguagem extremamente popular, é a mais usada para escrita de aplicações de sistema.

45 Linguagem de programação desenvolvida por Bjarne Stroustrup em 1983, baseada em C e acrescentando funcionalidade de orientação a objeto a ela.

funcionalidades de orientação a objeto, embora haja perda de performance. Existem ainda emuladores que usam linguagens de execução mais lenta, como Java[46], C#[47] e até Visual Basic[48], embora estes sejam exceções.

No caso do C, diversas otimizações podem ser feitas. O compilador geralmente possui opções para otimização do código gerado, e estas otimizações devem ser ativadas. Existem certas técnicas de otimização específicas para emulação de processadores, memória, placas gráficas, etc. Estas otimizações serão discutidas nas seções apropriadas deste capítulo.

3.5.1 Overhead

Um dos maiores problemas de performance nos emuladores é o *overhead*. *Overhead* é o excesso de instruções de máquina geradas, de forma transparente ao programador, no início e no final de cada função. Estas instruções são colocadas lá pelo compilador para executar a troca de contexto, ou seja, salvar os registradores, o endereço da pilha e outros dados para que o processador entre "limpo" na nova função.

A Figura 8 ilustra isto. Do lado esquerdo, há uma simples função em C que retorna a soma de dois números. No lado direto, está o código assembly gerado pelo compilador (no caso, o gcc[49]). Pode-se ver que o compilador gerou 6 instruções assembly para esta função. Duas delas preparam o processador para entrar na função (*overhead* superior), duas executam a operação e duas preparam o processador para retornar à função principal (*overhead* inferior).

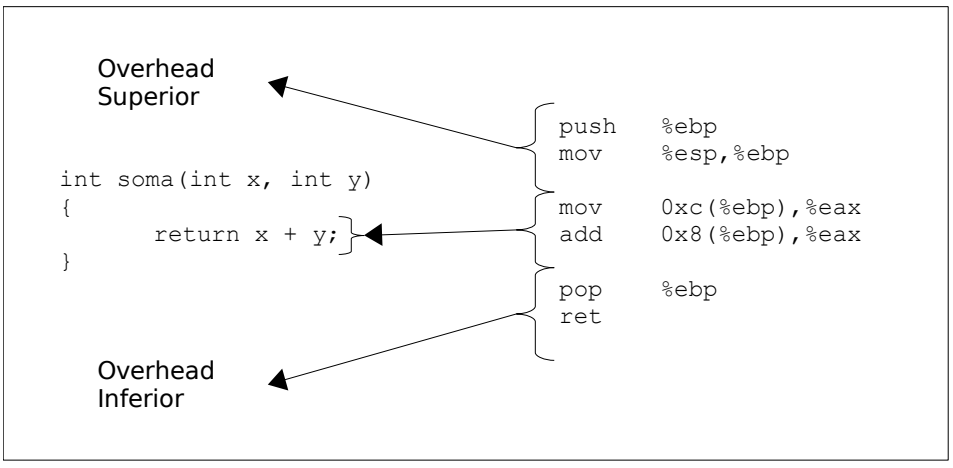


Figura 8 - *Overhead* em uma função C.

46 Linguagem de programação desenvolvida pela Sun Microsystems no começo dos anos 90, orientada a objetos e voltada para execução em uma máquina virtual, permitindo assim uma portabilidade completa.
47 Linguagem de programação desenvolvida pela Microsoft, baseada em C++ e no Java.
48 Linguagem de programação desenvolvida pela Microsoft, baseada no Basic e ligada a uma ferramenta de desenvolvimento RAD própria.
49 Conjunto de compiladores de código aberto produzido pelo Projeto GNU e distribuído pela Free Software Foundation. Do inglês *GNU Compiler Collection*, que significa Coleção de Compiladores GNU.

Na maioria dos programas, isto não é um problema, pois as funções são longas e não são chamadas com tanta freqüência. Num emulador, entretanto, o *overhead* é um problema sério, pois as funções são chamadas milhares (ou até milhões) de vezes por segundo. No exemplo acima, dois terços das instruções são gastos em *overhead*. Se tal tempo for perdido na emulação com o overhead, o emulador pode sofrer de séria perda de performance.

Uma das soluções propostas é a programação diretamente em assembly. Neste caso, o programador tem o controle de quanto *overhead* é realmente necessário, e pode se livrar daquilo que não é necessário. O grande problema da programação assembly, como já foi citado antes, é a falta de portabilidade.

Outra solução proposta é uso de funções *inline*. Neste caso, ao invés do compilador transformar uma função C em uma subrotina assembly, ele substitui a chamada da função pela própria função. Neste caso não há overhead, pois para o programa é como se o código da função chamada tivesse sido inserido dentro da função chamadora. A desvantagem deste método é que, se não for usado com cuidado, pode fazer com que o programa cresça demais, especialmente no caso de funções *inline* que são chamadas de vários lugares no programa.

CONCLUSÃO

REFERÊNCIAS BIBLIOGRÁFICAS

ALONI, Dan. Cooperative Linux. **Proceedings of the Linux Symposium**, Ottawa, v. 1. 2004.

BORIS, Daniel. **How Do I Write An Emulator?**. 1999.. Disponível em <http://personals.ac.upc.edu/vmoya/docs/HowToDanBoris.txt>. Acesso em 28 ago. 2006.

BRITISH Computer Society. **A Glossary of Computing Terms**. 10ª ed. Boston: Addison Wesley, 2002. 390 p.

CLD. **TONC: Regular Tiled Backgrounds**. 2006. Disponível em <>. Acesso em 09 Out. 2006.

DEL BARRIO, Victor Moya. **Study of the Techniques for Emulation Programming**. Barcelona: 2001. 152 p. (Tese de graduação) Facultat d'Informàtica de Barcelona. Disponível em <http://personals.ac.upc.edu/vmoya/docs/emuprog.pdf>. Acesso em 15 ago. 2006.

FAYZULLIN, Marat. **HOWTO: Writing a Computer Emulator**. 1997. Disponível em <http://fms.komkon.org/EMUL8/HOWTO.html>. Acesso em 15 Out. 2006.

FAYZULLIN, Marat. **fMSX: Portable MSX Emulator**. . Disponível em <http://fms.komkon.org/fMSX/>. Acesso em 27 ago. 2006.

GILHEANY, Steve. **Evolution of Intel Microprocessors 1971 to 2007**. 2006. Disponível em <http://www.cs.rpi.edu/~chrisc/COURSES/CSCI-4250/SPRING-2004/slides/cpu.pdf>. Acesso em 11 set. 2006.

INTEL Corporation. **IA-32 Intel Architecture Software Developer's Manual:** Volume 2A: Instruction Set Reference, A-M. 2006. Disponível em <ftp://download.intel.com/design/Pentium4/manuals/25366620.pdf>. Acesso em 11 set. 2006.

ISRAEL, Kirk. **2600 101: Into the Breach**. 2004. Disponível em <http://

alienbill.com/2600/101/02breach.html>. Acesso em 24 set. 2006.

JACOBS, Andrew John. **6502 Instructions**. 2002. Disponível em <http://www.obelisk.demon.co.uk/6502/>. Acesso em 12 set. 2006.

KELLER, Robert M. **Computer Science:** Abstraction to Implementation. Harvey Mudd College: 1997. Disponível em <http://www.cs.hmc.edu/claremont/keller/webBook/ch01/sec01.html>. Acesso em 09 set. 2006.

KORTH, Martin. **Atari 2600 Specifications**. 2006. Disponível em <http://nocash.emubase.de/2k6specs.htm>. Acesso em 20 set. 2006.

KORTH, Martin. **Pan Docs (Gameboy Specifications)**. 2001. Disponível em <http://nocash.emubase.de/pandocs.htm>. Acesso em 20 set. 2006.

LAUREANO, Marcos. **Máquinas Virtuais e Emuladores**. 1ª ed. São Paulo: Novatec, 2006. 184 p.

LINDHOLM, Tim; YELLIN, Frank. **The Java Virtual Machine**. 2ª ed. Boston: Addison Wesley, 1999. 473 p.

MICROSOFT Corporation. **Memory parity errors: Causes and suggestions**. 2006. Disponível em <http://support.microsoft.com/?kbid=101272>. Acesso em 20 set. 2006.

MONTEIRO, Mário A. **Introdução à Organização de Computadores**. 3ª ed. Rio de Janeiro: LTC, 1996. 397 p.

MURDOCCA, Miles J. e HEURING Vincent P. **Introdução à Arquitetura de Computadores**. 1ª ed. Rio de Janeiro: Campus, 2000. 512 p.

OLIVEIRA, Rômulo Silva de et al. **Sistemas Operacionais**. 2ª ed. Porto Alegre: Sagra Luzzatto, 2001. 247 p.

PATTERSON, David A. e HENNESSY, John L. **Organização e Projeto de Computadores**. 3ª ed. São Paulo: Elsevier, 2005. 484 p.

TANENBAUM, Andrew S.. **Organização Estruturada de Computadores**. 3ª ed. Rio de Janeiro: Prentice/Hall do Brasil, 1990. 460 p.

VMWARE Inc. **Virtualization Overview: VMWare Whitepaper**. . Disponível em <http://www.vmware.com/pdf/virtualization.pdf>. Acesso em 15 ago. 2006.

WEBER, Raul Fernando. **Fundamentos de Arquitetura de Computadores**. 1ª ed. Porto

Alegre: Sagra Luzzatto, 2000. 262 p.

WIKIPEDIA, The Free Encyclopedia. **Framebuffer**. 2006d. Disponível em <http://en.wikipedia.org/w/index.php?title=Framebuffer&oldid=76765516>. Acesso em 14 Out. 2006.

WIKIPEDIA, the Free Encyclopedia. **Vectrex**. 2006c. Disponível em <http://en.wikipedia.org/w/index.php?title=Vectrex&oldid=76443485>. Acesso em 24 set. 2006.

WIKIPEDIA, the Free Encyclopedia. **Cathode Ray Tube**. 2006b. Disponível em <http://en.wikipedia.org/w/index.php?title=Cathode_ray_tube&oldid=75878917>. Acesso em 20 set. 2006.

WIKIPEDIA, the Free Encyclopedia. **Clock Signal**. 2006a. Disponível em <http://en.wikipedia.org/w/index.php?title=Clock_signal&oldid=72530327>. Acesso em 28 ago. 2006.

WRIGHT, Steve. **Stella Programmer's Guide**. 1979. Disponível em <http://alienbill.com/2600/101/docs/stella.html>. Acesso em 21 set. 2006.

ULTRAHLE: Technical Information. 2006. Descreve informações técnicas do emulador UltraHLE. Disponível em <http://www.emuunlim.com/UltraHLE/old/techinfo.htm>. Acesso em 27. ago 2006.