ADRIANO BEDESCHI DE SOUZA
EDUARDO HISASHI SATO

EMULAÇÃO DO GAMEBOY E GAMEBOY COLOR PARA A PLATAFORMA PC/WINDOWS

Projeto de Formatura apresentado à disciplina PCS 2050 – Laboratório de Projeto de Formatura II, da Escola Politécnica da Universidade de São Paulo.

Área de Concentração:
Engenharia de Computação

Orientador:
Prof. Dr. Francisco Enéas da Cunha Lemos

São Paulo
2004

AGRADECIMENTOS

Ao orientador Prof. Dr. Francisco Enéas da Cunha Lemos, pela confiança apresentada na equipe.

A todos que, direta ou indiretamente, colaboraram no desenvolvimento deste trabalho.

RESUMO

O trabalho apresenta a construção de um emulador em software do videogame portátil GameBoy, e sua versão com cores, GameBoy Color, criados pela Nintendo. Para o desenvolvimento do mesmo foram estudados métodos de emulação e arquitetura de computadores. O texto apresenta os métodos e procedimentos utilizados para a emulação do portátil, cujo hardware é descrito detalhadamente. Os dados contidos no texto são frutos de um difícil trabalho de procura por informações sobre o funcionamento do GameBoy e seus variantes.

ABSTRACT

This project presents the writing of a software-based Gameboy emulator, and its color version, Gameboy Color, both of which are handheld videogame systems created by Nintendo. In order to write the emulator, the authors had to study emulation and computer architecture. This document presents methods and procedures that were used on the handheld's emulation, the hardware of which is thoroughly described on this text. The data contained herein was gained through a hard work of searching for information on the workings of the Gameboy System and its variants.

SUMÁRIO

LISTA DE FIGURAS

LISTA DE ABREVIATURAS E SIGLAS

GB – GameBoy.

GBP – GameBoy Pocket.

GBC – GameBoy Color.

GBA – GameBoy Advance.

ROM – *Read Only Memory*, normalmente utilizada para representar os jogos do GameBoy.

RAM – *Random Access Memory*.

VRAM – Video RAM.

OAM – *Object Attribute Memory*.

MBC – *Memory Bank Controller*.

TDT – *Tile Data Table Register*.

TIMA – *Timer Counter Register*.

TMA – *Timer Modulo Register*.

TAC – *Timer Control Register*.

IF – *Interrupt Flag Register*.

LCDC – LCD *Control Register*.

STAT – LCDC Status *Register*.

DMA – *Direct Memory Access*.

IE – *Interrupt EnableRegister*.

LISTA DE SÍMBOLOS

$ - Símbolo utilizado para representar números hexadecimais. Exemplo: $FF

LISTA DE TABELAS

1 INTRODUÇÃO

Este tópico apresenta o objetivo do trabalho, a motivação e justificativa para a realização do mesmo, e a organização do documento, com a descrição de seus principais capítulos.

1.1 Objetivo

O objetivo do projeto foi o desenvolvimento de um emulador, em software, do videogame portátil GameBoy, e sua versão com cores, GameBoy Color, comercializados pela Nintendo. O emulador deve aceitar os mesmos dados, executar os mesmos programas e produzir os mesmos resultados que o computador emulado. Desse modo, com o programa desenvolvido é possível jogar imagens de jogos (ROM's) feitos para GameBoy em um computador IBM PC com sistema operacional Windows.

1.2 Motivação

O motivo da escolha deste projeto pela equipe é o de que o mesmo permitiu aprendizado nas seguintes áreas de computação:

- Emulação: Entender como um emulador, tão utilizado hoje em dia, funciona. Esta foi a principal razão para a equipe.

- Arquitetura e organização de computadores: A arquitetura do GameBoy teve que ser estudada e entendida. Nisso inclui-se o processador, a entrada e saída, os periféricos, o mapeamento de memória etc. Além disso, foi fundamental conhecer bem a linguagem assembly da CPU emulada. Durante o desenvolvimento também foram trabalhados conceitos de organização computacional como interrupções, DMA, *timing* dos dispositivos e outros.

- Arquitetura do IBM PC: Como o projeto foi implementado totalmente em linguagem assembly, a arquitetura do PC, características e conjunto de instruções dos processadores x86 foram estudados e entendidos com bom grau de profundidade.

- Otimização de código: Uma das vantagens que a linguagem assembly proporciona é o alto desempenho dos programas produzidos. Porém, desenvolver programas em assembly sem o devido cuidado ou atenção gera programas com desempenho igual ou muitas vezes pior que programas gerados por compiladores com otimização. Durante o desenvolvimento do projeto foram estudadas algumas técnicas de otimização em assembly, levando em consideração a arquitetura dos PC's e o funcionamento interno dos processadores da família IA-32 e compatíveis.

O estudo das áreas acima citadas representou grande parte do desafio para o desenvolvimento do projeto, pois o profundo entendimento de vários aspectos das mesmas foi necessário para que alcançássemos sucesso no desenvolvimento do sistema.

1.3 Organização do documento

Este documento está organizado da seguinte forma: No capítulo 2 (Emulação) é discutido o conceito de emulação, e são apresentados os métodos de emulação mais utilizados. No capítulo 3 (Nintendo GameBoy) o portátil emulado é descrito minuciosamente. No capítulo 4 (Especificação do Projeto) são apresentados os requisitos funcionais e não funcionais do emulador, além da descrição de sua estrutura. No capítulo 5 (Metodologia) são discutidas as características e peculiaridades do projeto e a conseqüência dos mesmos no processo de desenvolvimento do emulador. No capítulo 6 (Projeto e Implementação) os algoritmos e idéias utilizados na implementação do trabalho são apresentados. No capítulo 7 (Testes e Avaliação), são exibidos os resultados da aplicação de alguns testes na versão final do emulador. No capítulo 8 (Considerações Finais) são

mostradas as conquistas que a equipe acredita ter alcançado, e também a opinião sobre futuros melhoramentos no projeto.

Existem três anexos no documento. O anexo A contém a lista de registradores de E/S do portátil e suas funcionalidades. O anexo B contém a lista de instruções do processador do GameBoy. O anexo C contém o cronograma final do projeto.

O único apêndice deste documento apresenta o CD com os arquivos do projeto.

2 EMULAÇÃO

2.1 Emulação e suas aplicações

Um emulador é um sistema que duplica todas as funcionalidades de outro sistema, permitindo que um computador de uma determinada plataforma execute programas escritos para uma plataforma diferente. O emulador deve aceitar os mesmos dados, executar os mesmos programas e produzir os mesmos resultados que o computador emulado.

As aplicações mais comuns de emuladores são:

- Emuladores podem servir como ferramenta de desenvolvimento de software. Ex: desenvolver programas para Palm/PocketPC, celulares, videogames etc em uma máquina Windows/Intel.

- Rodar programas ou jogos escritos para rodar em hardware que não se encontra mais disponível no mercado.

Além disso, muitos emuladores existentes, além de serem capazes de rodar programas de outras plataformas fielmente, apresentam recursos extras não existentes na plataforma emulada. Exemplos:

- Ferramentas de debug (depuração): Janelas que exibem informações sobre o estado atual da máquina emulada, tais como: o conteúdo dos registradores da CPU, o estado atual dos dispositivos de E/S, o código atualmente em execução desmontado (*disassembled*) etc. Podem estar presentes também controles de fluxo de programa (*breakpoints*, *step-in*, *step-over*, *step-out* etc)

- *Savestates*: Muitos emuladores permitem salvar o estado atual da máquina emulada em arquivos, de tal forma que seja possível recuperá-lo mais tarde.

- Filtros de imagem: Podem suavizar a imagem original gerada pela máquina emulada e simular o aspecto de uma tela de TV.

Existem centenas de *sites* na Internet dedicados à emulação. A grande maioria dos computadores dos últimos 20 anos já foi emulada de alguma maneira. Esse crescente interesse nesse assunto é devido ao fato de que o grande poder de processamento disponível nos PC's atuais tem permitido a talentosos programadores produzirem emuladores para máquina complexas, que rodam em tempo real e permitem às pessoas executarem programas ou jogos que jamais estariam disponíveis nos PC's.

Um emulador é um programa extremamente desafiante para se escrever (admitindo emulação em software). A sua construção envolve uma grande quantidade de pesquisa aliada a um certo grau de iniciativa necessária para preencher os vazios na informação disponível. Também é necessário o completo entendimento do hardware a ser emulado, bem como habilidade para construir os algoritmos, muitas vezes complexos, a serem utilizados na emulação.

2.2 Métodos de emulação

Existem basicamente três métodos de emulação:

2.2.1 Recompilação estática

Neste caso, o emulador traduz o código binário do programa para o código de máquina da plataforma desejada. O programa resultante poderá ser rodado diretamente em um PC, por exemplo, sem o uso de outro programa. Apesar deste

método ter uma boa performance, há casos em que é impossível utilizá-lo. Um exemplo é o caso de programas que se auto modificam.

2.2.2 Recompilação dinâmica

Este método é basicamente o mesmo que o anterior. Mas com recompilação dinâmica, ao invés de traduzir todo o programa de uma só vez, o processo é feito por partes durante a execução do mesmo cada vez que uma instrução de JUMP ou CALL é encontrada. Esse método pode ser combinado com o de recompilação estática.

2.2.3 Emulação interpretativa

O emulador lê o código binário do programa, o decodifica e executa as ações apropriadas sobre os registradores, sobre a memória e sobre os dispositivos de E/S emulados. As vantagens desse método são: facilidade para depuração e facilidade para sincronizar todos os dispositivos emulados. A desvantagem é que este método toma bastante tempo de processamento.

Este foi o método de emulação escolhido para o projeto. Primeiro pela facilidade de implementação. Segundo porque para obter alta compatibilidade (rodar tantos jogos quanto for possível) é necessário que a sincronização da CPU com os periféricos seja perfeita, o que é difícil de se conseguir com os outros dois métodos. Terceiro porque o poder de processamento dos PC's atuais é ordens de grandeza maior que a máquina a ser emulada, sendo possível assim, através deste método, emular um GameBoy rodando em sua velocidade nativa sem problemas.

O *loop* principal de emulação interpretativa de um videogame pode ser representado pelo fluxograma abaixo:

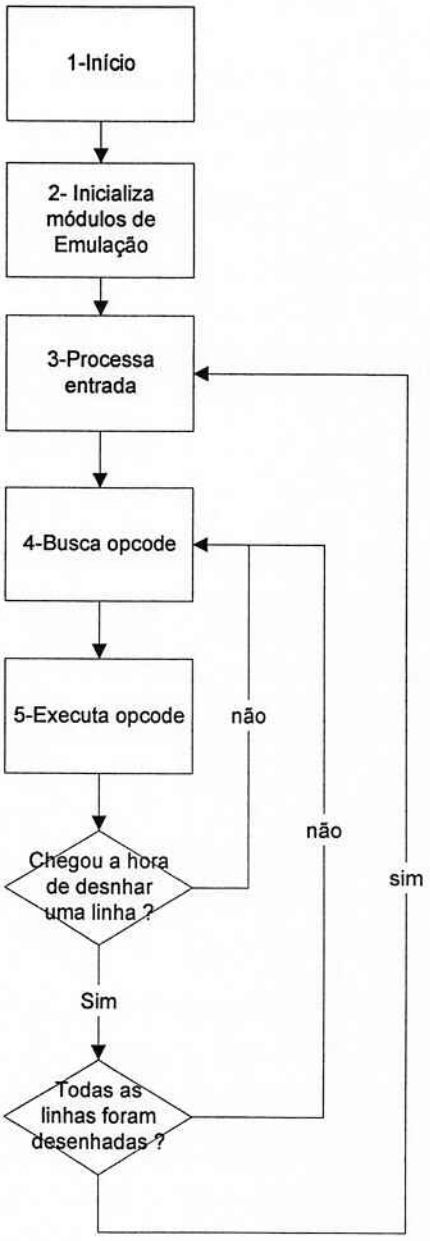

Figura 1 – *Loop* principal de um emulador interpretativo

O passo 2 acima inicializa as variáveis globais do programa, além de ajustar o conteúdo dos registradores da CPU, os dispositivos de E/S, e as memórias em seus valores iniciais. O passo 3 verifica a entrada do usuário, o que significa verificar se o usuário pressionou alguma tecla. Em seguida uma estrutura de dados representando os controles do videogame emulado é preenchida de acordo.

Um programa de computador é composto por código binário (código de máquina). Tal código por sua vez é composto por uma seqüência de *opcodes*, onde cada *opcode* representa uma instrução de CPU. A CPU do GameBoy, por exemplo, tem instruções com *opcodes* que variam em tamanho de 1 a 4 bytes.

Um emulador interpretativo funciona de modo muito semelhante a um processador real. O passo 4 do fluxograma acima lê o próximo *opcode* a ser executado do código binário do programa. O passo 5 executa a instrução representada pelo *opcode* buscado anteriormente, atualizando o estado da CPU emulada.

3 NINTENDO GAMEBOY

3.1 História do GameBoy

O GameBoy é um videogame portátil projetado e produzido pela Nintendo. Ele possuiu uma tela LCD em preto e branco capaz de exibir quatro níveis de cinza, uma CPU Z80 modificada rodando a 4Mhz e 4 canais de som. Seus jogos são armazenados em chips ROM, os quais são colocados em placas de circuito e embalados em cartuchos plásticos para proteção. Os cartuchos são mais caros de se produzir do que mídias magnéticas ou CD's, porém são muito mais robustos e fáceis de se utilizar.

A versão original foi lançada em 1988 para competir com produtos rivais de outras empresas de jogos eletrônicos. Devido ao seu preço relativamente baixo e à grande gama de jogos disponíveis, o GameBoy rapidamente dominou o mercado. Hoje em dia seu irmão mais novo, e muito mais poderoso, o GameBoy Advance, ainda domina o mercado de videogames portáteis.

O GameBoy passou por duas transformações e melhoramentos no decorrer de sua estadia no mercado. A primeira alteração veio com o lançamento do GameBoy Pocket, o qual não continha diferenças na especificação do hardware, mas era bem menor, e possuía uma tela melhor (menos *ghosts*) e utilizava apenas duas baterias (ao invés das quatro do GameBoy normal). A segunda modificação no portátil, e a mais significativa, veio com o lançamento do GameBoy Color, com capacidade de exibição de 64 cores simultaneamente. O GBC é compatível com as versões anteriores, porém a velocidade da CPU e o tamanho da memória de vídeo foram dobradas.

Figura 2 – GameBoy e GameBoy Color

3.2 Descrição do funcionamento e hardware do GameBoy

Os dados apresentados nesta seção foram obtidos através de diversos documentos retirados do *site* Devrs [5], conversas com outros autores de emuladores, e testes realizados pela própria equipe. Nesta seção são citados diversos registradores de E/S presentes no GameBoy. A lista desses registradores e suas respectivas funções podem ser encontradas no anexo A.

A figura abaixo mostra os principais componentes de hardware do GameBoy

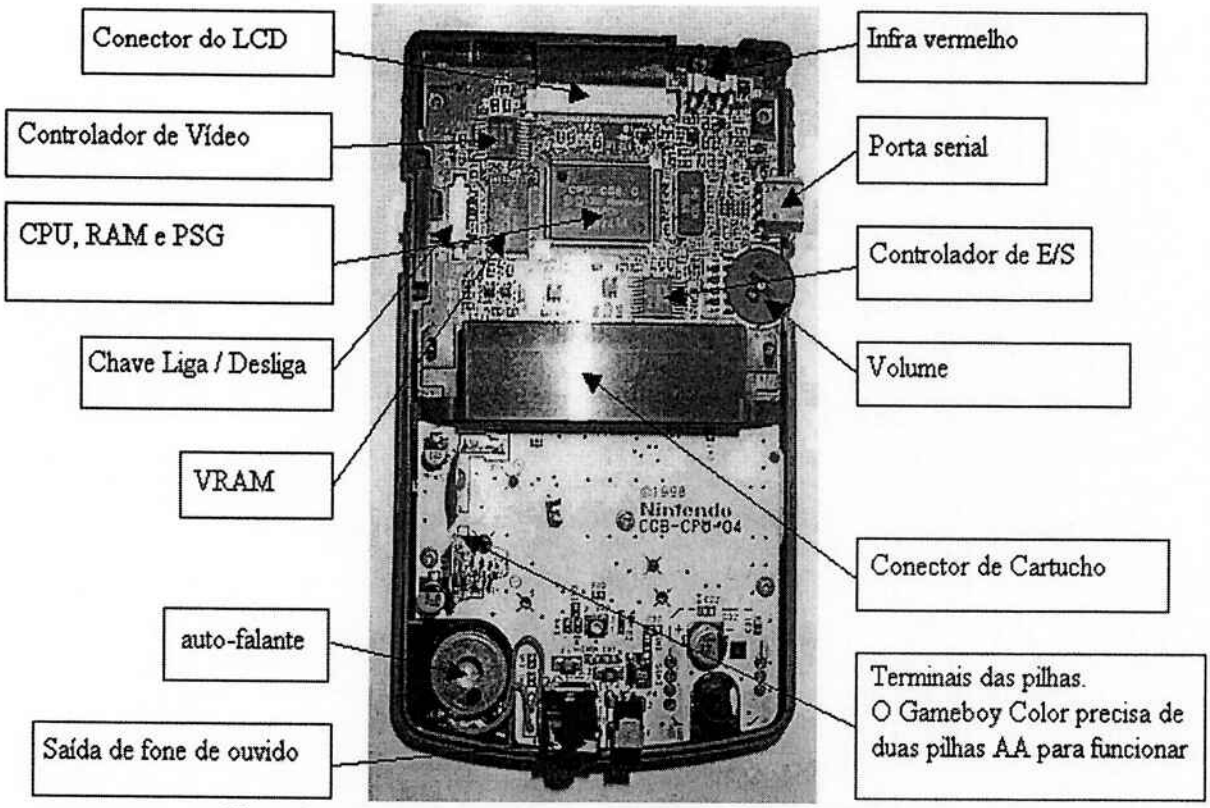

Figura 3 – Principais componentes de hardware do GameBoy

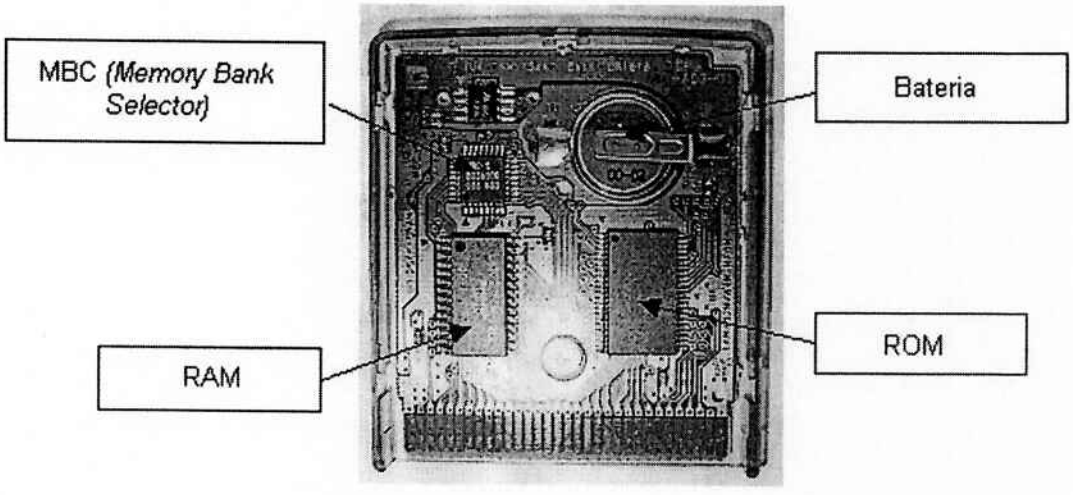

Figura 4 – Principais componentes de hardware do GameBoy (cartucho)

3.2.1 Resumo das características do hardware

A tabela a seguir exibe as principais características do hardware do GameBoy.

Componente / Característica	Valor
CPU	8 bits (similar ao Z80)
Velocidade do *clock*	4,194304 MHz (8,388 MHz para o GBC)
Memória principal	8KB
Memória de vídeo (VRAM)	8KB
Sincronismo horizontal	9198 KHz
Sincronismo vertical	59,73 Hz
Tamanho da tela	2,6 polegadas
Cores	4 níveis de cinza (32768 cores para o GBC)
Resolução	160x144 pixels (20x18 *tiles*)
Número máximo de *sprites*	40
Número máximo de *sprites* por linha	10
Tamanho dos *sprites*	8x8 e 8x16 pixels
Som	4 canais com som pseudo-estéreo

Tabela 1 – Características do hardware do GameBoy

3.2.2 O processador (CPU)

A CPU do GameBoy é um processador similar ao Z80 da Zilog. Trata-se de um processador de 8 bits com *databus* de 16 bits. Apesar de rigorosamente a CPU do GameBoy não ser um Z80, referiremos a ele como tal. A descrição completa do Z80 e de suas instruções do Z80 encontra-se em seu manual do usuário [1].

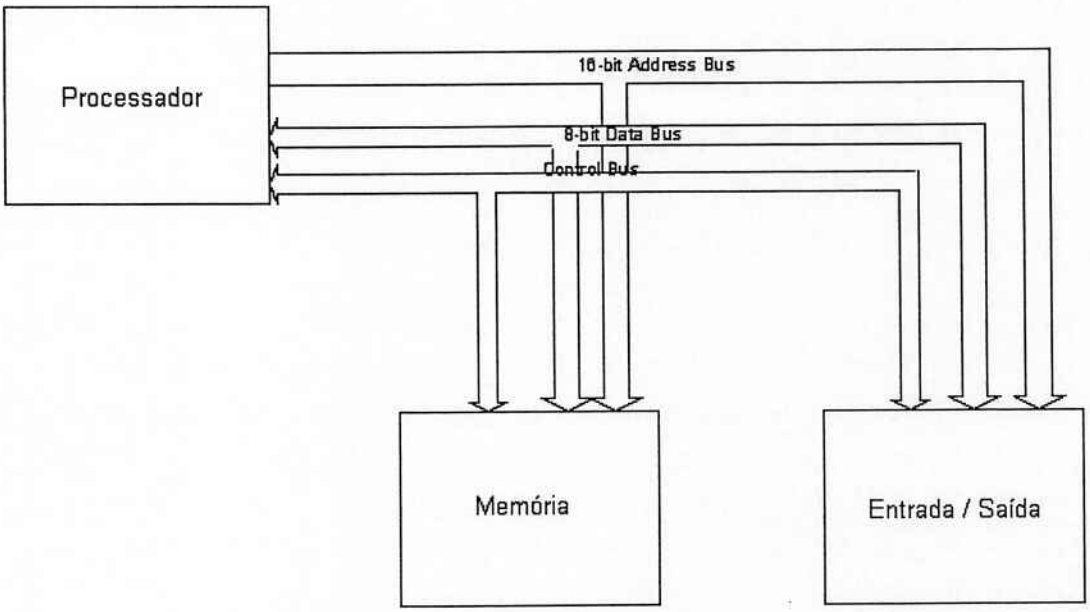

Figura 5 – Z80 CPU

Os registradores do Z80 são:

Registrador(es)	Descrição
A	Acumulador (8 bits)
F	Registrador de *flags* (8 bits)
B, C, D, E, H, L	Registradores de uso geral (8 bits)
SP	Registrador de pilha (16 bits)
PC	*Program Counter* (16 bits)

Tabela 2 – Registradores do Z80

O registrador F do processador do GameBoy não utiliza as *flags* de paridade e *overflow*. As flags existentes são as seguintes:

- Z – Ativo caso o resultado da última operação é zero.
- N – Ativo caso a última operação foi uma subtração.
- H – Ativo caso tenha ocorrido um *carry-out* na metade menos significativa dos valores utilizados na última operação (bit 3).
- C – Ativo caso tenha ocorrido um *carry-out* na última operação.

Também podemos acessar os registradores de 8 bits como registradores de 16 bits.

Registrador(es)	Descrição
BC, DE, HL	Registradores de uso geral (16 bits)

O Conjunto de instruções da CPU do GameBoy é o mesmo do Z80, com, com apenas algumas inclusões, exclusões e alterações de *opcode*.

As instruções adicionais em relação ao Z80 são as seguintes.

Opcode (0x)	Instrução	Descrição
E8 nn	ADD SP, nn	nn = *offset* de 8 bits
22	LDI (HL), A	Escreve A em (HL) e incrementa HL
32	LDD (HL), A	Escreve A em (HL) e decrementa HL
2A	LDI A, (HL)	Escreve (HL) em A e incrementa HL
3A	LDD A, (HL)	Escreve (HL) em A e decrementa HL
F0 nn	LD A, ($FF00 + nn)	Utilizado para acessar os registradores de E/S (serão explicados mais adiante)

F2	LD A, ($FF00 + C)	---
E0 nn	LD ($FF00 + nn), A	---
E2	LD ($FF00 + C), A	---
08 bb aa	LD (aabb), SP	Escreve SP em (aabb)
F8 nn	LD HL, SP + nn	nn = *offset* de 8 bits
10	STOP	Pára o processador e a tela até que um botão seja pressionado
CB 3A	SWAP A	Troca *nibbles* mais e menos significativos do acumulador

Tabela 3 – Instruções adicionais do processador do GameBoy em relação ao Z80

As instruções excluídas em relação ao Z80 são as seguintes.

Instrução
Toda instrução que utiliza os registradores IX e IY
Toda instrução IN / OUT
Toda instrução EX
Toda instrução com *opcode* iniciado por ED
Toda instrução de pulo, chamada e retorno condicionais em relação às *flags* de paridade e *overflow*, inexistentes no processador do GameBoy

Tabela 4 – Instruções excluídas do processador Z80

As instruções com *opcodes* alterados são as seguintes.

Instrução	***Opcode* no Z80 (0x)**	***Opcode* no GB (0x)**
LD A, (aabb)	3A bb aa	FA bb aa
LD (aabb), A	32 bb aa	EA bb aa
RETI	ED 4D	D9

Tabela 5 – Instruções com *opcodes* alterados em relação ao Z80

O número de *clocks* utilizados por cada instrução do processador do GameBoy é diferente do apresentado pelo Z80 normal. A lista das instruções do processador com os respectivos timings está no anexo B.

3.2.3 O mapeamento de memória

O mapeamento da memória no GameBoy é apresentado na tabela abaixo, são 64KB endereçados.

Memória	Endereço (0x)	Notas
Interrupt Enable Register	FFFF	Escritas neste local habilitam e desabilitam determinadas interrupções.
RAM interna de pilha	FF80-FFFE	Utilizada pela pilha.
Portas de E/S	FF00-FF7E	Escritas nesta área configuram dispositivos de E/S. Leituras recebem dados provindos dos mesmos.
Sprite Attribute Memory (OAM)	FE00-FEFF	Dados sobre os *sprites* são armazenados nesta região de memória.
Cópia da memória principal	E000-FDFF	RAM principal também pode ser escrita e lida através desta área.
Memória principal – 8KB RAM (x8 no GBC)	C000-DFFF	RAM principal. Paginável apenas no GBC.
8KB RAM paginável	A000-BFFF	Área na qual são paginados blocos de memória RAM presentes em alguns cartuchos.
8KB Video RAM (x2 no GBC)	8000-9FFF	Memória de vídeo. Paginável apenas no GBC.
16KB ROM paginável	4000-7FFF	Área na qual são paginados blocos de memória ROM provindos dos cartuchos.
16KB ROM bloco #0	0000-3FFF	Área reservada para os primeiros 16KB do jogo.

Tabela 6 – Mapeamento da memória do GameBoy

3.2.4 Áreas reservadas no primeiro bloco de ROM

A seguir é apresentado o *header* que todo cartucho possui, situado no primeiro *bank* da ROM. Essa área é reservada.

Endereço (0x)	Descrição
0000	RST $00 chama este endereço.
0008	RST $08 chama este endereço.
0010	RST $10 chama este endereço.
0018	RST $18 chama este endereço.
0020	RST $20 chama este endereço.
0028	RST $28 chama este endereço.
0030	RST $30 chama este endereço.
0038	RST $38 chama este endereço.
0040	Interrupção de varredura vertical.
0048	Interrupção de status do LCDC.
0050	Interrupção de *overflow* do *Timer*.
0058	Interrupção de comunicação serial.
0060	Interrupção P10 –P13.
0100 - 0103	Ponto inicial de execução de um ROM. Normalmente aqui são encontrados instruções NOP e JP.
0104 – 0133	Nintendo logo (jogos não rodam se estiver errado): CE ED 66 66 CC 0D 00 0B 03 73 00 83 00 0C 00 0D 00 08 11 1F 88 89 00 0E DC CC 6E E6 DD DD D9 99 BB BB 67 63 6E 0E EC CC DD DC 99 9F BB B9 33 3E
0134 – 0142	Título do jogo em ASCII em letras maiúsculas. Se for menor que 16 bytes, o restante é preenchido com $00's.
0143	$80 significa que o jogo é para GBC, $00 significa que o jogo é para qualquer outro variante do GB.
0144	*Nibble* mais significativo do código de licença.
0145	*Nibble* menos significativo do código de licença.
0146	Indicador de GB e SGB. $00 indica GameBoy e $03 indica

	que são utilizadas funções do Super GameBoy.
0147	Tipo de cartucho: $00 – ROM apenas $01 – ROM + MBC1 $02 – ROM + MBC1 + RAM $03 – ROM + MBC1 + RAM + BATT $05 – ROM + MBC2 v06 – ROM + MBC2 + BATTERY $08 – ROM + RAM $09 – ROM + RAM + BATTERY $0B – ROM + MMM01 $0C – ROM + MMM01 + SRAM $0D – ROM + MMM01 + SRAM + BATTERY $0F – ROM + MBC3 + TIMER + BATTERY $10 – ROM + MBC3 + TIMER + RAM + BATTERY $11 – ROM + MBC3 $12 – ROM + MBC3 + RAM $13 – ROM + MBC3 + RAM + BATTERY $19 – ROM + MBC5 $1A – ROM + MBC5 + RAM $1B – ROM + MBC5 +RAM + BATTERY $1C – ROM + MBC5 + RUMBLE $1D – ROM + MBC5 + RUMBLE + SRAM $1E – ROM + MBC5 + RUMBLE + SRAM + BATTERY $1F – Procket Camera $FD – Bandai TAMA5 $FE – Hudson HuC-3 $FF – Hudson HuC-1
0148	Tamanho da ROM: $00 – 256 Kb = 32 KB = 2 *banks* $01 – 512 Kb = 64 KB = 4 *banks* $02 – 1 Mbit = 128 KB = 8 *banks*

	$03 – 2 Mbit = 256 KB = 16 *banks* $04 – 4 Mbit = 512 KB = 32 *banks* $05 – 8 Mbit = 1 MByte = 64 *banks* $06 – 16 Mbit = 2 MByte = 128 *banks* $52 – 9 Mbit = 1,1 MByte = 72 *banks* $53 – 10 Mbit = 1,2 MByte = 80 *banks* $54 – 12 Mbit = 1,5 MByte = 96 *banks*
0149	Tamanho da RAM: $00 – Sem RAM $01 – 16 Kb = 2 KB = 1 *bank* $02 – 64 Kb = 8 KB = 1 *bank* $03 – 256 Kb = 32 KB = 4 *banks* $04 – 1 Mbit = 128 KB = 16 *banks*
014A	Código de destino: $00 – Japão $01 - Internacional
014B	Código de licença: $33 – Checar 0144 e 0145 para obter o código $79 – Accolade $A4 - Konami
014C	$00
014D	Valor de checagem complementar (jogos não rodam se estiver errado).
014E – 014F	*Checksum* (byte mais significativo primeiro) produzido pela adição de todos os bytes do cartucho exceto os dois bytes de *checksum*. O *checksum* será então os dois bytes menos significativos do valor calculado.

Tabela 7 – Áreas reservadas no primeiro bloco de ROM dos cartuchos.

3.2.5 Tipos de cartucho

A seguir são apresentados os tipos de cartucho para GameBoy, indicados pelo byte localizado no endereço 0x0147, como mostrado anteriormente. Os cartuchos com mais de 32KB precisam do chip MBC para mapear os blocos da ROM.

Note que a área de memória 0000-7FFF é usada tanto para ler a ROM, como para escrever nos registradores de controle dos MBC's, como será mostrado a seguir.

3.2.5.1 MBC1

0000-3FFF - ROM Bank $00 (Leitura apenas)
Esta área sempre contém os primeiros 16KB da ROM (primeiro bloco).

4000-7FFF - ROM Bank $01 - $7F (Leitura apenas)
Esta área pode conter qualquer bloco a partir do segundo bloco da ROM, permitindo o endereçamento de até 125 blocos (quase 2 Mbytes). Como descrito abaixo, os blocos de número $20, $40 e $60 não podem ser utilizados, resultando nos 125 blocos possíveis.

A000-BFFF - RAM Bank $00 - $03, se necessário (Leitura e Escrita)
Esta área é usada para endereçar memória RAM do cartucho, se existir. Esta RAM externa é normalmente acompanhada de um bateria, permitindo armazenar no cartucho posições de jogos ou pontuações, mesmo após o GameBoy ter sido desligado.

0000-1FFF - RAM Enable (Escrita apenas)
Para acessar a memória RAM do cartucho, é preciso antes habilitá-la escrevendo nesta área da memória. Escrevendo um valor com 0Ah nos 4 bits menos significativos habilita a RAM, qualquer outro valor a desabilita.

2000-3FFF - ROM Bank Number (Escrita apenas)

Escrevendo nesta área de memória seleciona os 5 bits menos significativos do número do bloco da ROM a ser mapeado. Quando $00 é escrito, o MBC traduz que o bloco $01 deve ser utilizado, não permitindo que o bloco 0 seja mapeado fora de sua área reservada (0000-3FFF). O mesmo acontece para os blocos $20, $40 e $60. Qualquer tentativa de endereçar estes blocos da ROM será redirecionada para os blocos $21, $41 e $61, respectivamente.

4000-5FFF - RAM Bank Number - or - Upper Bits of ROM Bank Number (Escrita apenas)

Escrevendo dois bits nesta área de memória pode-se selecionar o bloco da RAM a ser mapeado na memória, ou então os dois bits podem representar os dois bits mais significativos do número do bloco da ROM a ser mapeado. O significado dos dois bits depende do modo selecionado (veja abaixo).

6000-7FFF - ROM/RAM Mode Select (Escrita apenas)

Indica o significado dos dois bits descritos acima. Escrevendo-se $00 nesta área de memória seleciona o ROM *Banking Mode*. Escrevendo-se $01 é selecionado o RAM *Banking Mode*. Quando o modo selecionado for o da ROM, o bloco da RAM mapeado deve ser o 0.

3.2.5.2 MBC2

0000-3FFF - ROM Bank $00 (Leitura apenas)

Igual ao MBC1.

4000-7FFF - ROM Bank $01-$0F (Leitura apenas)

Igual ao MBC1, porém apenas o mapeamento dos 16 primeiros blocos é suportado.

A000-A1FF – 512 x 4 bits RAM, embutido no chip MBC2 (Leitura e Escrita)

O MBC2 não suporta RAM externa, ao invés disso ele já possui embutido no seu chip 512 x 4 bits de RAM. Como os dados dessa RAM consistem apenas de 4 bits, apenas os 4 bits menos significativos são utilizados ao escrever ou ler nela.

0000-1FFF - RAM Enable (Escrita apenas)

Escrevendo nesta área, como no MBC1, habilita ou desabilita a RAM do MBC2. O bit menos significativo do byte mais significativo do endereço precisa ser 0 para habilitar ou desabilitar a RAM do MBC2. Por exemplo, os seguintes endereços podem ser utilizados: 0000-00FF, 0200-02FF, 0400-04FF, ..., 1E00-1EFF. A área de memória sugerida é 0000-00FF.

2000-3FFF - ROM Bank Number (Escrita apenas)

Escrevendo um valor (XXXXBBBB – X = qualquer, B = bits para seleção do bloco) em 2000-3FFF selecionará um bloco da ROM a ser mapeado em 4000-7FFF. O bit menos significativo do byte mais significativo do endereço precisa ser 1 para selecionar o bloco da ROM. Por exemplo, os seguintes endereços podem ser utilizados: 2100-21FF, 2300-23FF, 2500-25FF, ..., 3F00-3FFF. A área de memória sugerida é 2100-21FF.

3.2.5.3 MBC3

Similar ao MBC1, exceto pelo fato de poder acessar todos os 16 Mbits da ROM sem necessidade de escrever em 4000-5FFF. Escrevendo um valor (XBBBBBBB – X = qualquer, B = bits para seleção do bloco) em 2000-3FFF selecionará um bloco da ROM a ser mapeado em 4000-7FFF.

O MBC3 possui um RTC (*Real Time Clock*) não encontrado em qualquer outro MBC.

3.2.5.4 MBC5

Similar ao MBC3, porém podendo acessar até 64 MBits de ROM e até 1 Mbit de RAM. Os 8 bits menos significativos dos 9 bits utilizados para selecionar o bloco são escritos em 2000-2FFF, enquanto que o bit mais significativo é escrito no bit menos significativo da área 3000-3FFF.
Escrevendo um valor (XXXXBBBB – X = qualquer, B = bits para seleção do bloco) em 4000-5FFF selecionará um bloco da RAM a ser mapeado em A000-BFFF.

O MBC5 foi criado para garantir a execução em dupla velocidade no GameBoy Color, mas os outros MBC's também rodam sem problemas com velocidade duplicada no GBC.

3.2.6 Seqüência de inicialização

Quando o GameBoy é ligado, um programa de 256 bytes no início da memória é executado. Este programa está localizado em uma ROM embutida no GameBoy.

A primeira coisa que o programa faz é ler as posições $104 - $133 do cartucho e colocar este logo da Nintendo na tela. Após isso, as posições $104 - $133 são novamente lidas, mas desta vez elas são comparadas com uma tabela interna mantida pelo GameBoy. Caso exista alguma diferença entres os dados, o GameBoy trava.

Em seguida o GameBoy calcula a soma dos bytes do cartucho e faz a comparação do *checksum*, como explicado anteriormente.

Em caso de sucesso nos testes, a ROM embutida no GameBoy é desativada e a execução do cartucho inicia na posição $100, com os seguintes valores iniciais para os registradores e para a pilha.

Registrador / Posição de memória	Valor Inicial
A	$01 para o GB $FF para o GBP $11 para o GBC
F	$B0
B	$00
C	$13
DE	$00D8
HL	$014D
SP	$FFFE
($FF05)	$00
($FF06)	$00
($FF07)	$00
($FF10)	$80
($FF11)	$BF
($FF12)	$F3
($FF14)	$BF
($FF16)	$3F
($FF17)	$00
($FF19)	$BF
($FF1A)	$7F
($FF1B)	$FF
($FF1C)	$9F
($FF1E)	$BF
($FF20)	$FF
($FF21)	$00
($FF22)	$00
($FF23)	$BF
($FF24)	$77
($FF25)	$F3

($FF26)	$F1
($FF40)	$91
($FF42)	$00
($FF43)	$00
($FF45)	$00
($FF47)	$FC
($FF48)	$FF
($FF49)	$FF
($FF4A)	$00
($FF4B)	$00
($FFFF)	$00

Tabela 8 – Valores iniciais dos registradores e pilha do GameBoy.

3.2.7 O controlador de vídeo – VDP (Vídeo Display Processor)

Os gráficos do GameBoy são compostos por 3 camadas; a camada de *background*, a camada de janela, e a camada de *sprites*. Em uma determinada parte de memória de vídeo (VRAM) são definidos caracteres de 8x8 pixels. No GameBoy normal cada pixel pode tem um de quatro 4 tons de cinza disponíveis, ou seja, são necessários 2 bits para representar cada pixel. Como cada caractere tem 8*8=64 pixels, cada caractere ocupa 16 bytes na VRAM. No caso do GameBoy Color, que possui o dobro de memória de vídeo em relação ao GameBoy normal, cada pixel pode ter até 8 cores.

A camada de *background* é composta por uma matriz de 32x32 caracteres de 8x8 pixels. Logo, as dimensões em pixels dessa camada é de 256x256. Porém, apenas um pedaço de tamanho 160x144 pixels dessa área de é mostrado no display. O programador pode escolher qual pedaço será exibido alterando o valor de registradores do VDP (registradores 0xFF42, *scroll* X, e 0xFF43, *scroll* Y). Esses registradores determinam a posição da área exibida, como mostra a figura abaixo.

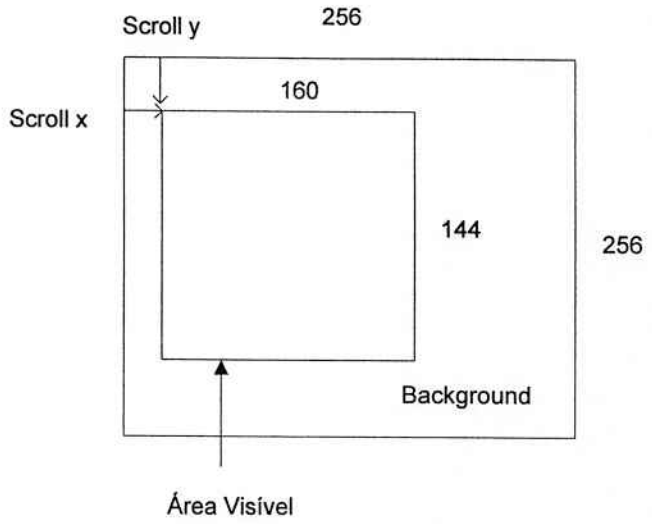

Figura 6 – *Background* e janela no GameBoy

Uma área da VRAM denominada *Background Tile Map* contém os caracteres a serem exibidos na tela pela camada de *background*. Esse mapa é organizado em 32 linhas de 32 bytes cada. Cada byte contém o número do caractere a ser exibido. Os caracteres são retirados do *Tile Data Table* (TDT), localizado em $8000-$8FFF ou $8800-$97FF. No primeiro caso, os *tiles* são numerados de 0 a 255 (ou seja, o caractere 0 fica na posição $8000). No segundo caso, os *tiles* são numerados de -128 a 127 (ou seja, o caractere 0 fica na posição $9000). O endereço do TDT a ser usado para a camada de background é selecionado através do registrador LCDC.

Existem dois diferentes *Background Tile Maps*. O primeiro está localizado em $9800-$9BFF. O segundo em $9C00-$9FFF. Apenas um deles pode estar ativo em um determinado momento. A escolha de qual deles deve estar ativo também é feita através do registrador LCDC.

A camada de janela é similar a camada de *background*. Ela é composta por uma matriz de 160x144 pixels. As coordenadas de posicionamento da janela (reguladas pelos registradores WNDPOSX e WNDPOSY) são dadas em relação à parte visível da camada de background. A janela pode ser ativada ou desativada através do registrador LCDC.

As imagens dos caracteres (*tiles*) são armazenadas na TDT como já foi mencionado. Cada imagem de 8x8 pixels ocupa 16 bytes, onde cada dois bytes representam uma linha.

Byte	Valor hexa	Valor binário
0	0xE7	11100111
1	0xBD	10111101

Tabela 9 – Exemplo de representação de uma linha de um caractere.

As cores correspondentes ao exemplo apresentado na tabela acima são obtidas da seguinte maneira.

Pixel	Byte 1	Byte 0	Valor do par	Cor
0	1	1	11	3
1	0	1	01	1
2	1	1	11	3
3	1	0	10	2
4	1	0	10	2
5	1	1	11	3
6	0	1	01	1
7	1	1	11	3

Tabela 10 – Cores obtidas no exemplo apresentado na tabela 9.

A cor pode então ser tirada da palheta, que é determinada pelo registrador BGP ($FF47).

Como foi dito anteriormente, existem duas TDT's, uma em $8000-8FFF, e outra em $8800-97FF. A primeira pode ser utilizada para os *sprites*, para o *background* e para a janela. Já a segunda não pode ser utilizada pelos *sprites*.

A camada de sprites é composta por caracteres de 8x8 ou 8x16 pixels. O VDP pode exibir até 40 *sprites*, porém, devido a limitação do hardware, apenas 10 *sprites* podem aparecer por linha. Os *sprites* possuem o mesmo formato que os caracteres, e os mesmo podem ser obtidos a partir do segundo TDT apenas. Os *sprites* possuem atributos, os quais estão localizados na *Sprite Attribute Table* (também denominado

Object Attribute Memory – OAM), que fica mapeado em $FE00-$FE9F. A OAM é dividida em 40 blocos de 4 bytes cada, onde cada bloco representa um dos 40 *sprites*.

A tabela a seguir mostra o significados dos 4 bytes de atributos de um *sprite*.

Byte	Descrição
0	Posição y na tela.
1	Posição x na tela.
2	Número do caractere.
3	*Flags:* • Bit 7 – Prioridade: Se este bit está zerado, o *sprite* é exibido acima do *background* e da janela. Se o bit está com valor 1, o *sprite* ficara escondido pelas cores 1, 2 e 3 do *background* e janela (o *sprite* sempre prevalece sobre a cor 0). • Bit 6 – *Flip* vertical. • Bit 5 – *Flip* horizontal. • Bit 4 – Número da palheta a ser utilizada. As cores do *sprite* são tiradas do registrador OBP1 caso este bit esteja com valor 1, e OBP0 caso contrário.

Tabela 11 – Atributos dos *sprites* no GameBoy.

3.2.8 Extensões do VDP no GameBoy Color

Existem basicamente duas diferenças entre VDP do GameBoy descrito acima e o VDP do GameBoy Color. Em primeiro lugar, o número de palhetas foi aumentado de 3 para 16, 8 para o *background* e 8 para o *foreground* (janela e *sprites*). A segunda diferença está no tamanho da VRAM, que foi dobrada no GBC.

3.2.9 Os dispositivos de E/S

Os três principais dispositivos do GameBoy são o controle, o PSG e o link serial. Todos funcionam através da escrita e leitura dos registradores de E/S localizados em

$FF00-$FF7E. A lista de todos os registradores e suas respectivas funções pode ser encontrada no anexo A.

3.2.10 O controlador de som – PSG (Programmable Sound Generator)

O controlador de som do GameBoy é composto por 4 canais de som pseudo-estéreo:

- Canal 1:

  - Produz ondas quadradas com *duty cycle.*variável, e funções *frequency sweep* e *envelope*. A função *frequency sweep* permite que se tenham efeitos de "portamento" nos quais a freqüência aumenta ou diminui durante o *playback*. A velocidade com a qual a freqüência aumenta ou diminui é controlável.

- Canal 2:

  - Produz ondas quadradas com *duty cycle* variável e função *envelope*. O canal 2 é idêntico ao canal 1, sem a função *frequency sweep*. A função *envelope* permite ter efeitos de *fade-in* e *fade-out,* nos quais o volume do som produzido aumenta ou diminui gradativamente. O canal possui 4 bits de resolução, de modo a poder produzir 16 diferentes níveis de amplitude.

- Canal 3:

  - Atua como um DAC (Digital-to-Analog Converter) de 4 bits que toca repetidamente um padrão de amostras de áudio. Esse padrão é definível pelo usuário.

- Canal 4:

  - Produz pseudo ruído branco com função *envelope*. O ruído é gerado por um contador polinomial, também conhecido como *Linear-Feeedback Shift Register* (LFSR). LFSRs são um tipo de contador binário que tem a característica especial de não contar na seqüência binária crescente normal.

3.2.11 Timer

O GameBoy possui um timer cuja freqüência pode ser 4096 Hz, 16384 Hz, 65536 Hz ou 262144 Hz. O valor da freqüência é selecionado pelo registrador TAC ($FF07). O registrador TIMA ($FF05) é incrementado com esta freqüência. Quando o mesmo sofre *overflow*, ele gera uma interrupção, e o seu valor é então reiniciado para o valor do registrador TMA (*Timer* Modulo).

3.2.12 Interrupções

A *flag* IME (*Interrupt Master Enable)* do processador do GB pode ser resetada pela instrução DI, proibindo a execução de qualquer tipo de interrupção. A instrução EI habilita novamente as interrupções, sendo que estas são reguladas pelo registrador IE e IF do GameBoy. As interrupções funcionam da seguinte forma:

Quando uma interrupção é gerada, a *flag* IF do processador é ativada.
Caso a *flag* IME estiver ativada e a *flag* do registrador IE correspondente à interrupção gerada também estiver ativada, as três ações seguintes são executadas.
- Desativa IME para prevenir novas interrupções.
- O PC (*Program Counter*) é colocado na pilha.
- É feito um pulo para o endereço da interrupção.

Ao retornar da interrupção, o registrador IF é zerado e a *flag* IME reativada.

3.2.12.1 Varredura vertical

Esta interrupção ocorre aproximadamente 60 vezes por segundo, sempre no início do período de varredura vertical. Durante este tempo, que dura aproximadamente 1,1

ms, o hardware de vídeo não está acessando a VRAM, estando portanto livre para ser acessado pelo programa.

3.2.12.2 LCDC Status

Existem várias razões para esta interrupção ocorrer, como descrito pelo registrador STAT ($FF40). Uma razão comum é para indicar ao usuário quando o VDP está para desenhar uma linha do LCD.

3.2.12.3 Timer overflow

Esta interrupção ocorre quando o registrador TIMA ($FF05) muda de $FF para $00.

3.2.12.4 High-to-low P10-P13

Esta interrupção ocorre na transição de qualquer uma das linhas de entrada do controle do GameBoy de 1 para 0, descrita pelo registrador P1 ($FF00).

4 ESPECIFICAÇÃO DO PROJETO

4.1 Especificação funcional

A função principal do emulador será rodar jogos comerciais do GameBoy e GameBoy Color na plataforma PC/Windows. O emulador deverá ter uma taxa de compatibilidade maior que 50%.

O emulador deve também disponibilizar diversos recursos não existentes no GameBoy.

As funções desempenhadas pelo sistema emulador são divididas nos seguintes grupos:

1. Funções de carregamento de programa e estado:
   1.1. Abrir ROM – carrega o programa no emulador e o executa.
   1.2. Fechar ROM – termina a execução do programa.
   1.3. Salvar estado – salva o estado atual do sistema emulador (valores dos registradores do cpu, memória etc).
   1.4. Carregar estado salvo – carrega o estado do sistema previamente salvo.

As funções para salvar e carregar estado (*save states*) são comumente utilizadas em emuladores de vídeo games para salvar o jogo permitindo que o usuário o continue posteriormente, ação q a maioria dos jogos não disponibiliza.

2. Funções de controle do CPU:
   2.1. Reiniciar CPU – reinicializa o sistema emulado.
   2.2. Pausar CPU – pausa a emulação.

3. Funções de controle de vídeo:
   3.1. Alterar modo de exibição:
      3.1.1. Exibição normal – apresenta a imagem de vídeo do GameBoy em seu tamanho real sem nenhuma escala ou filtro aplicado.
      3.1.2. Exibição em tamanho dobrado – apresenta a imagem com o dobro do tamanho original sem aplicação de filtro.
      3.1.3. Exibição em tamanho dobrado com *scanlines* – apresenta a imagem com o dobro do tamanho original e com *scanlines* na mesma, simulando o aspecto de uma tela na TV.

   3.2. Alterar palheta de vídeo:
      3.2.1. Utilização da palheta normal – utiliza a palheta do GameBoy normalmente.
      3.2.2. Seleção manual das cores da palheta (GameBoy normal apenas) – permite que o usuário altere as 4 cores da palheta do GameBoy normal.
      3.2.3. Utilização da palheta em tons de cinza (GameBoy Color apenas) – altera as cores da palheta para tons de cinza.
      3.2.4. Utilização da palheta negativa (GameBoy Color apenas) – altera as cores da palheta para suas complementares.

   3.3. Ativar ou desativar camadas de imagem:
      3.3.1. Camada de *Background* – permite ativar ou desativar a exibição da camada de *background* na imagem final.
      3.3.2. Camada de *Sprites* – permite ativar ou desativar a exibição da camada de *sprites* na imagem final.
      3.3.3. Camada de Janela - permite ativar ou desativar a exibição da camada de janela na imagem final.

   3.4. Eliminar limite de *sprites* por *scanline* do hardware do GameBoy – permite exibir mais do que 10 *sprites* por *scanline*.

4. Funções de controle de som:
   4.1. Ativar ou desativar som:
      4.1.1. Canal de onda quadrada I – permite ativar ou desativar a ação do canal no som final emulado.
      4.1.2. Canal de onda quadrada II – permite ativar ou desativar a ação do canal no som final emulado.
      4.1.3. Canal de *wave* – permite ativar ou desativar a ação do canal no som final emulado.
      4.1.4. Canal de ruído branco – permite ativar ou desativar a ação do canal no som final emulado.

5. Funções de debug do sistema - Janelas que exibem informações sobre o estado atual da máquina emulada:
   5.1. *Disassembler* – possibilita a visualização em linguagem assembly do programa carregado no emulador, assim como a execução passo a passo do programa, com atomicidade representada por uma instrução da CPU.
   5.2. *Header Viewer* – possibilita a visualização do cabeçalho do programa carregado no emulador.
   5.3. *Memory Viewer* – possibilita a visualização da memória do GameBoy enquanto o programa carregado é executado.
   5.4. *Map Viewer* – possibilita a visualização gráfica da parte da memória de vídeo destinada a guardar o background da imagem.
   5.5. *Tile Viewer* - possibilita a visualização gráfica da parte da memória de vídeo destinada a guardar os *tiles* disponíveis para a construção da imagem.
   5.6. OAM Viewer - possibilita a visualização gráfica dos *sprites* correntes de acordo com sua tabela de definição, assim como seus atributos.

4.2 Requisitos não funcionais

1. O emulador deve apresentar uma interface simples para o usuário, permitindo a ativação e alteração de parâmetros dos recursos disponíveis pelo sistema. A entrada de controle do GameBoy será emulada através de seu mapeamento para o teclado.

2. O emulador deve apresentar uma performance elevada, podendo ser executado em velocidade máxima mesmo em computadores antigos como Pentium's 100 MHz.

3. O programa deve ser robusto, para que não trave caso sejam abertos arquivos de ROM's defeituosas ou arquivos que não sejam ROM's.

4.3 Plataforma

O emulador será desenvolvido em linguagem Assembly para processadores Intel da família IA-32. A família IA-32 inclui os processadores x86 lançados pela Intel desde o Pentium até o Pentium IV.
O emulador deve rodar em computadores com os sistemas operacionais Windows 98, Windows NT, Windows2000, Windows XP e Windows 2003.

4.4 Estrutura do emulador

O emulador foi divido de acordo com o diagrama abaixo:

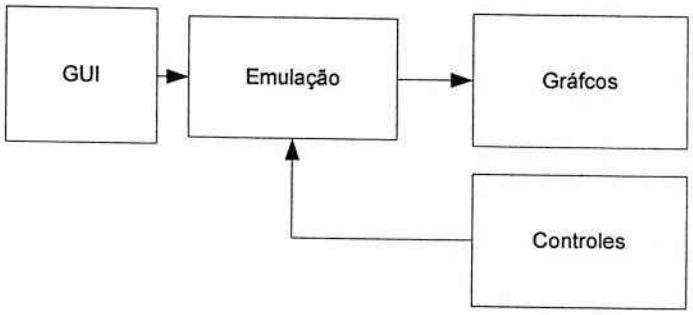

Tabela 12 – Estrutura do emulador

- GUI – *Graphical User Interface*: É a interface com o usuário. É composta de janelas gráficas e das rotinas que controlam a lógica de interação com o usuário.
- Emulação: Esta camada implementa o *loop* principal da emulação interpretativa.
- Gráficos: Essa camada é responsável por exibir o buffer de display na tela e aplicar os filtros de imagem
- Controles: Controla os dispositivos de entrada (no caso o teclado).

A camada de emulação será dividida em módulos correspondentes aos componentes do hardware do GameBoy. Cada módulo será desenvolvido procurando-se separar o máximo possível a lógica dos componentes do hardware emulado.

Abaixo estão apresentados os principais componentes da camada de emulação. A subdivisão desta camada nos módulos finais é apresentada no capítulo 6 (Projeto e Implantação).

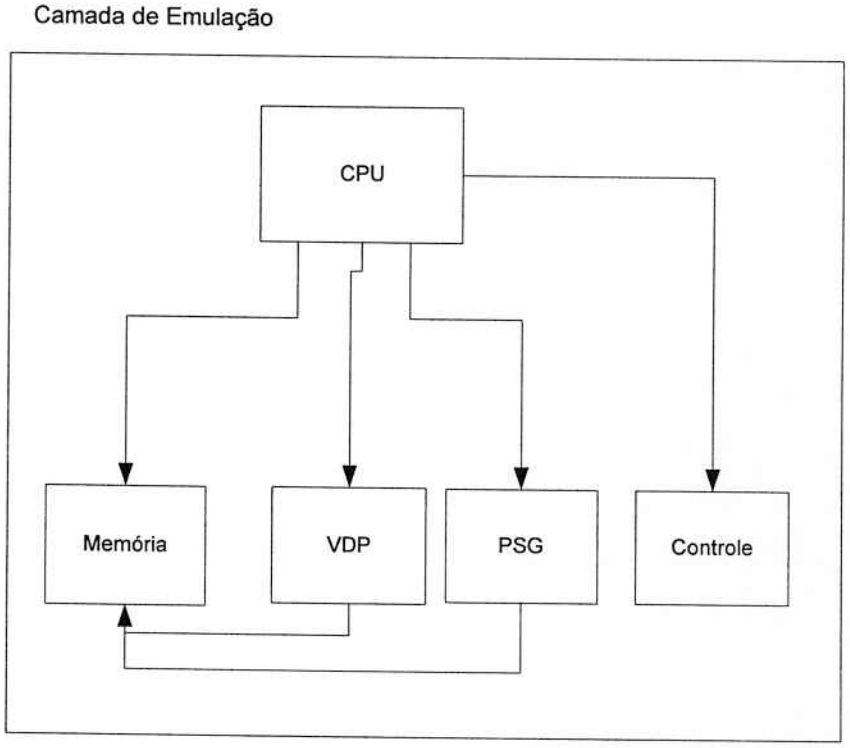

Tabela 13 – Camada de emulação

A natureza do projeto do emulador faz com que a dificuldade do mesmo não esteja na sua estruturação, e sim em sua implementação.

5 METODOLOGIA

Como a equipe consta de apenas dois integrantes, todo o processo que envolveu o estudo de tecnologias e estudo do GameBoy, assim como o planejamento e posterior implementação dos módulos do projeto, foi efetuado por ambos os membros em sua totalidade.

O projeto, apesar de ter como produto final um software, é fruto de um extenso estudo baseado em hardware, tanto do sistema a ser emulado, como do sistema emulador. Esta característica do trabalho fez com que partes do software fossem remodeladas e novamente programadas constantemente.

A divisão de tarefas para a implementação do sistema não pôde ser feita de modo muito modular, pois neste projeto específico, cada módulo do sistema é muito dependente dos outros, ou seja, foi preciso muitas vezes que ambos os membros da equipe trabalhassem em conjunto com o mesmo módulo. Além disso, o perfeito entendimento de como foi programado cada módulo foi sempre necessário para o desenvolvimento dos módulos seguintes. Um fato importante é que a escolha pela implementação do emulador totalmente em assembly eliminou a possibilidade de utilização de orientação a objetos para o desenvolvimento do emulador.

Com relação aos testes efetuados, cabe ressaltar que o único módulo capaz de ser testado separadamente foi o processador emulado. Todos os outros módulos não puderam ser testados individualmente, pois são interdependentes, e os mesmo praticamente não funcionam sem a existência dos outros.

Com os fatos apresentados, pode-se concluir que nenhuma metodologia especifica foi utilizada na construção do emulador. As características apresentadas pelo projeto o tornam peculiar quanto ao processo de sua construção.

6 PROJETO E IMPLEMENTAÇÃO

6.1 Considerações gerais

O projeto foi implementado em linguagem assembly para a família x86 e compatíveis. O montador (*assembler*) utilizado foi o Masm 6.14 da Microsoft.

Não foi utilizada nenhuma biblioteca extra, fora a API do Windows.

6.2 A divisão em módulos

O projeto foi feito em linguagem assembly, que não requer que os programas sejam estruturados tampouco sigam o paradigma da orientação de objetos. Por isso o programa foi estruturado em sub-rotinas (funções) e módulos (conjuntos de funções, macros e variáveis globais).

Os módulos foram divididos em basicamente dois grupos: o primeiro é o dos módulos que lidam diretamente com emulação, e o segundo são os módulos restantes, que lidam entre outras coisas com criação e gerenciamento de janelas, caixas de diálogo, multimídia etc.

Cada módulo é composto fisicamente de um ou mais arquivos fonte de extensão ".ASM" e um ou mais arquivos de inclusão com extensão ".INC".

Os módulos escolhidos foram os seguintes:

6.2.1 Módulos de emulação

Módulo	Arquivo fonte	Arquivo de inclusão	Função
Controls	Controls.asm	Controls.inc	Emula o joypad (controle) do gameboy
Emulator	Emulator.asm	Emulator.inc	Principal módulo do programa. Contém o loop de emulação.
Z80Core	Z80Core.asm, Instset.asm	Instset.inc, Opcode.inc	Emulação da CPU e funções correlatas.
Memory	Memory.asm	Memory.inc	Mapeia a memória. Lida com entrada e saída.
VDP	VDP.asm	VDP.inc	Módulo de emulação do VDP do Gameboy.

Tabela 14 – Módulos de emulação

6.2.2 Outros módulos

Módulo	Arquivo fonte	Arquivo de inclusão	Função
Disassembler	Disassembler.asm	Disassembler.inc, mnemonics.inc, mnemonicsCB.inc	Caixa de diálogo com funções de debug.
Display	Display.asm	Display.inc	Contém funções para exibir as imagens geradas na emulação na tela.
Header Viewer	HeaderViewer.asm	HeaderViewer.inc	Caixa de diálogo com informações sobre o cartucho carregado.
Logger	Logger.asm	Logger.inc	Caixa de Diálogo e funções para exibir informações e auxiliar na depuração do programa.
Main	Main.asm		Contém o ponto de entrada do programa.
Map Viewer	MapViewer.asm	MapViewer.inc	Caixa de diálogo com informações sobre a tabela de caracteres do VDP.
Memory Viewer	MemoryViewer.asm MemoryViewerControl.asm	MemoryViewer.inc MemoryViewerControl.inc	Caixa de diálogo que exibe conteúdo da memória mapeada no Gameboy.
OAM Viewer	OAMViewer.asm	OAMViewer.inc	Caixa de diálogo com informações sobre a tabela de atributos de sprites do VDP.

Sound	Sound.asm	Sound.inc	Funções para gerar som pela placa de som do computador.
TileViewer	TileViewer.asm WakaTileControl.asm	TileViewer.inc WakaTileControl.inc	Caixa de diálogo com informações sobre os caracteres definidos na VRAM.

Tabela 15 – Módulos de gerenciamento

6.3 A emulação da CPU

A emulação do processador principal (CPU) consiste basicamente em executar instruções a partir do código de máquina do programa a ser executado. Isso significa ler seqüencialmente bytes do programa (*fetch*) e executar os *opcodes* (instruções) que eles representam (execute).

Essas duas ações são implementadas pela rotina `ExecuteOpcode` do módulo Z80Core e por um conjunto de funções que contém uma rotina para cada instrução do conjunto de instruções do Z80. Abaixo a macro que implementa as instruções da forma LD r, r´:

```
; LD r, r'
LD_r1_r2 macro opcode:REQ, reg1:REQ, reg2:REQ

inst&opcode:

      ifdifi <reg1>, <regA>
            mov         al, reg2
            mov         reg1, al
      else
            mov         dh, reg2
      endif

      inc         esi                     ; PC++

      ret
endm
```

Figura 7 – Z80.asm – Macro que implementa as instruções na forma LD r, r´

A macro acima é expandida uma vez para cada combinação possível de r e r´, onde r, r´ ∈ {A, B, C, D, E, H, L}.

Um exemplo de instrução um pouco mais complexa é a instrução ADD A, X:

```
; ADD
DO_ADD macro

      xor         ecx, ecx
      xor         dl, dl

      mov         bh, dh
      add         dh, bl

      setz  ah

      adc         ecx, ecx
      shl         ecx, 4
      or          dl, cl      ; seta flag CF

      xor         bh, bl
      xor         bh, dh

      ror         ah, 1
      or          dl, ah      ; seta flag Z

      and         bh, 010H
      shl         bh, 1
      or          dl, bh      ; seta flag H
endm
```

Figura 8 – Z80.asm – Macro que implementa as instruções na forma ADD A, X

Portanto, para cada uma das 512 instruções do Z80 temos uma macro semelhante às mostradas acima, que quando chamadas são expandidas em funções que executam as instruções.

A rotina `ExecuteOpcode` é responsável por pegar o código de máquina do programa em execução, decodificar cada instrução do programa, calcular o endereço da função que implementa essas instruções e chamar a função adequada. Esta rotina, além disso retorna o número de ciclos gastos pela CPU para executar as instruções. Esse valor é utilizado no *loop* de emulação (que será abordado adiante) para sincronizar os dispositivos de Entrada/Saída, de tal modo que chamadas às rotinas de emulação do VDP ou PSG, por exemplo, sejam feitas nos momentos corretos. Além disso, o número de ciclos gastos é fundamental para executar as mudanças de modo associadas ao controlador de vídeo. Há que se destacar que esse sincronismo entre os diversos dispositivos emulados tem um grande impacto na compatibilidade do software rodado pelo emulador.

Em pseudocódigo:

```
while(programaRodando) {

      proximo_opcode = ReadMemory(PC);

      endereco_funcao = vetorDeEnderecos[proximo_opcode];

      ciclos_gastos = ChamaFuncaoDeImplementação(endereco_funcao);

      PC = PC + 1;
      return ciclos_gastos;
}
```

Figura 9 – Pseudocódigo da rotina `ExecuteOpcode`

E a rotina propriamente dita:

```
ExecuteOpcode:

      test  reg_HALT, 1
      jz          _no_HALT

      ;AQUI "HALT 1"
      mov         eax, 4
      ret

_no_HALT:

      ; Fetch
      movzx       esi, regPC          ; esi <- PC
      mov         edi, esi            ; edi <- esi

      call  ReadMemory                ; eax <- opcode

      movzx       ebp, OPSTATES[eax]
      movzx       edx, regAF                ; edx <- AF

      ; Execute
      ; chama rotina de execução do opcode
      call  dword ptr[_switch_OpCode + 4*eax]

      mov         regAF, dx           ; atualiza globais
      mov         regPC, si

      ; retorna numero de ciclos gastos
      mov         eax, ebp
      ret
```

Figura 10 – Rotina `ExecuteOpcode`

Para poder executar as instruções, além dessas funções são definidas variáveis globais que guardam o conteúdo dos registradores da CPU. As macros de implementação de instruções mostradas acima executam suas ações alterando os valores dessas variáveis.

Existe uma variável para cada registrador:

```
.data

regAF       LABEL word
regF        db 0B0H
regA        db 001H

regBC       LABEL word
regC        db 013H
regB        db 000H

regDE       LABEL word
regE        db 0D8H
regD        db 000H

regHL       LABEL word
regL        db 04DH
regH        db 001H

regPC       dw 00100H
regSP       dw 0FFFEH
```

Figura 11 – Z80.asm –Variáveis globais que representam os registradores da CPU emulada

6.4 A emulação do VDP

A emulação do VDP (controlador de vídeo) é tratada no módulo VDP, composto pelos arquivos VDP.asm e VDP.inc. Esse módulo contém rotinas que transformam o conteúdo da memória de vídeo (VRAM mapeada em $8000 - $9FFF) em uma imagem de 160 linhas que é mostrada na tela.

A imagem formada é composta de 3 camadas: background, *window* (janela) e *sprites*. Tais camadas têm que ser renderizadas levando se em conta aspectos como as prioridades e, a inversão no eixo X e Y dos caracteres. Esses atributos são configurados em bytes situados dentro da VRAM.

O módulo VDP possui uma rotina que renderiza cada uma das 3 camadas. Cada uma dessas rotinas desenha por vez uma linha das 160 linhas que compõem cada quadro de emulação. Elas recebem como parâmetro o número da linha a ser desenhada.

As rotinas são as seguintes:

- `DrawScanline`
- `DrawWindow`
- `DrawScanlineSprites` / `DrawScanlineSprites16`

A função dessas rotinas é basicamente interpretar o conteúdo da VRAM e gerar valores RGB correspondentes.

Todos os quadros de emulação (são produzidos 60 quadros por segundo) são desenhados primeiramente num buffer, para depois serem jogados para a tela. Essa técnica é chamada *Double Buffer* é utilizada para evitar que seja possível enxergar a imagem se formando na tela e para conseguir uma maior fluidez e velocidade na animação. As rotinas enumeradas logo acima desenham no *double buffer*. A rotina

que desenha o *double buffer* na tela chama-se `RefreshScreen` e encontra-se no módulo Display.

O conteúdo do registrador LCDC (LCD *Control*) determina vários aspectos da configuração do controla dor de vídeo. O bit 2 desse registrador determina se os *sprites* terão tamanho de 8x8 pixels ou 8x16 pixels. No primeiro caso a rotina `DrawScanlineSprites` deve ser chamada, no segundo a rotina `DrawScanlineSprites16` deve ser chamada para renderizar os *sprites*.

Para se gerar uma *scanline* completa com as três camadas precisamos chamar três das funções acima mencionadas na seguinte ordem: primeiro DrawScanline, em seguida DrawWindow e por último DrawScanlineSprites, ou DrawScanlineSprites16. Isto é feito no *loop* de emulação. Abaixo o trecho do *loop* que faz chamada a esse módulo:

```
; Desenha Scanline
                pushad
                call  DrawScanline
                mov         al, 020H
                test  reg_LCDC, al
                jz          __no_window
                call  DrawWindow
__no_window:
                mov         al, 04H
                test  reg_LCDC, al
                jz          __8x8
                call  DrawScanLineSprites16
                jmp         __done_render_scanline
__8x8:          call  DrawScanLineSprites
__done_render_scanline:
                popad
```

Figura 12 – Emulator.asm – Parte do loop de emulação que faz chamada às rotinas do módulo VDP

6.5 A implementação do PSG

A implementação do PSG é tratada no módulo PSG. A principal funcionalidade desse módulo, que é geração de amostras de áudio na freqüência de 44100Hz.

A maior dificuldade na emulação do chip provém do fato do mesmo funcionar a uma freqüência diferente e que não é nem múltipla nem um divisor das freqüências de saída das placas de som dos PC's (131072 Hz). Assim várias conversões e ajustes têm que ser feitos para que a emulação possa produzir 44100 amostras de áudio por segundo. O estado interno do PSG é mantido em variáveis globais, com valores convertidos com base 44100.

A primeira rotina desse módulo é a `PSGInit`. Esta rotina calcula os períodos de tempo 1/256 s, 1/128 s e 1/64 s em termos de amostras a 44100Hz e armazena esses valores em variáveis globais. Esses valores são utilizados nas funções de *envelope, sound sweep* e *sound length* dos canais de áudio do PSG.

A rotina `PSG_IO` administra as escritas aos registradores do PSG ($FF10 a $FF40). A rotina executa os cálculos necessários de conversão de freqüências, de acordo com o registrador escrito e armazena o resultado dos cálculos nas variáveis globais de estado.

A rotina `PSG_Write_Samples`, gera amostras de áudio à freqüência de 44100Hz e as escreve num buffer. Esse buffer é passado periodicamente para a placa de som.

6.6 A implementação do acesso à memória

O acesso à memória no emulador deve se comportar exatamente como ocorre num GameBoy real. Tal comportamento está implementado no módulo Memory através das rotinas de leitura `ReadMemory`, `ReadMemoryW`, e das rotinas de escrita `WriteMemory` e `WriteMemoryW`. Todos os acessos à memória dentro do emulador são feitos através dessas rotinas.

Essas rotinas emulam o funcionamento dos chips MBC e a maneira como eles mapeiam os chips de memória no espaço endereçado pelo processador.

Além das rotinas de leitura e escrita, o módulo Memory possui rotinas para alocação e liberação de memória para o carregamento de cartuchos (`MemAlloc` e `MemFree`), possui uma função para carregamento de imagem de cartucho a partir de disco (`LoadFile`) e uma função que inicializa o sistema de mapeamento de memória (`MemoryMapInit`).

A rotina `MemoryMapInit` descobre a partir da imagem de jogo carregada qual dos chips MBC o cartucho que contém o jogo possui. Isso é feito analisando-se o byte de endereço $0147 do chip de memória ROM do cartucho. Abaixo um trecho da rotina que faz isso:

```
        mov         ebx, cartROM
        mov         al, [ebx + CARTRIDGE_TYPE_OFFSET]

        cmp         al, 00h
        jnz         @F
        mov         mapperType, NO_MAPPER
        mov         hasRAM, 0
        mov         hasBatery, 0
        jmp         _done_set_mapper

@@:     cmp         al, 01h
        jnz         @F
        mov         mapperType, MBC1
        mov         hasRAM, 0
        mov         hasBatery, 0
        jmp         _done_set_mapper

@@:     cmp         al, 02h
        jnz         @F
        mov         mapperType, MBC1
        mov         hasRAM, 1
        mov         hasBatery, 0
        jmp         _done_set_mapper

; etc ....
```

Figura 13 – Memory.asm – Parte da rotina MemoryMapInit que descobre qual chip MBC utilizar.

Uma vez inicializado o mapeamento de memória, as rotinas de leitura e escrita ao serem chamadas se comportarão de acordo com o chip MBC detectado.

Quando o programa tenta escrever um byte em algum endereço situado entre $0000 e $8000, endereços nos quais os bancos de memória enxergados são configurados, a rotina `WriteMemory` se reconfigura de tal modo que subseqüentes leituras feitas através da rotina `ReadMemory` retornem o conteúdo da memória referente ao banco selecionado.

Isso é feito através de apontadores. Por exemplo, se alterarmos o número do banco que é enxergado nas posições de endereço entre $4000 e $7FFF para o banco 3 do cartucho, fazemos o apontador apontar para esse banco:

Pseudocódigo:

```
offset = TAMANHO_BANCO * numero_do_banco;
apontador = &(ROM[0]) + offset;
```

Assim, quando for feita uma leitura nas posições de endereço entre $4000 e $7FFF, faz-se:

Pseudocódigo:

```
return apontador[endereço - $4000];
```

Além de implementar o comportamento dos chips MBC, as rotinas `ReadMemory` e `WriteMemory` implementam a comunicação com dispositivos de E/S que são os *timers*, o controlador (*gamepad*) e o acesso ao DMA.

6.7 A implementação do *loop* de emulação

O *loop* de emulação é a parte mais importante do emulador. Ele é implementado pela rotina `EmulatorRunDebug`. Ele é responsável por acionar todas os módulos de emulação do emulador de maneira sincronizada. O fluxograma abaixo ilustra o comportamento do *loop*:

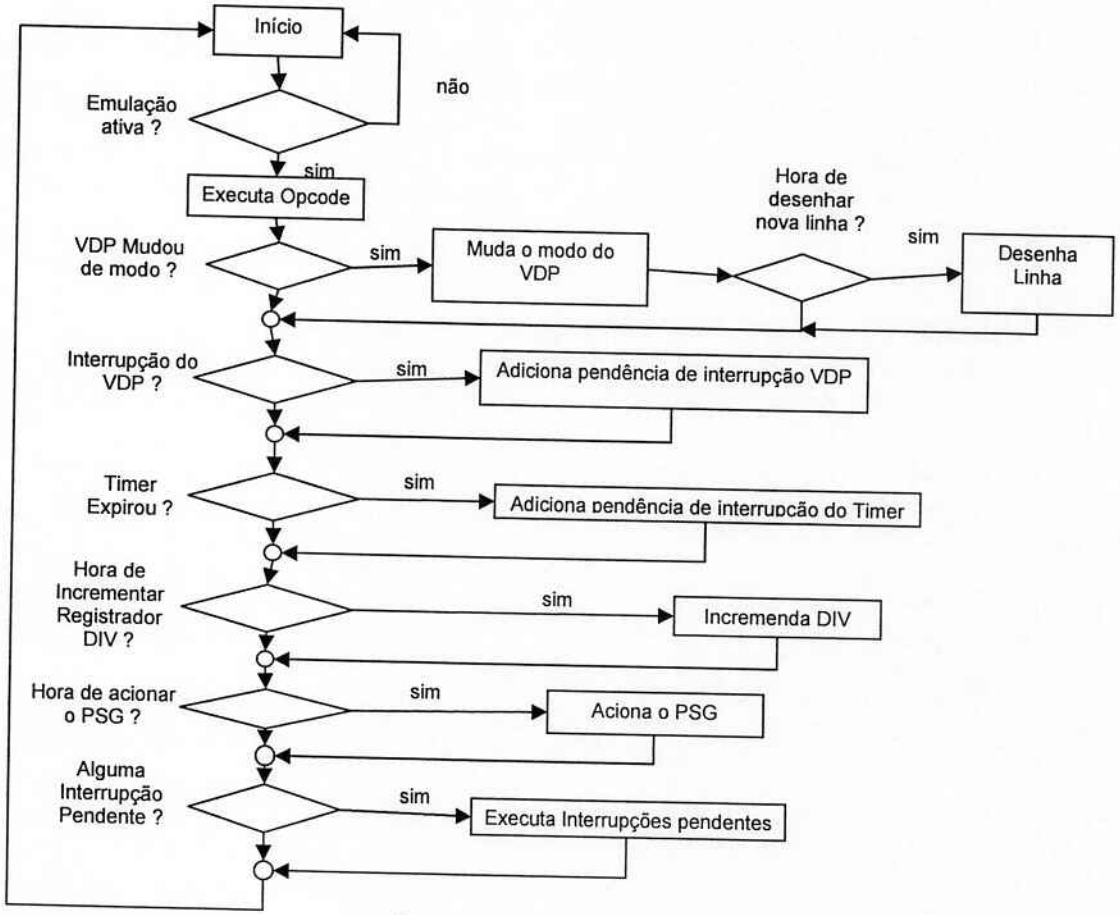

Figura 14 – *Loop* de emulação.

6.8 Módulo Controls

Esse módulo possui apenas uma função: `UpdateControls`. Essa rotina emula o *gamepad* do gameboy, de modo que o teclado do PC funcione como controlador do emulador. Os botões do *gamepad* estão mapeados para teclas do teclado da seguinte maneira:

- cima: seta para cima
- baixo: seta para baixo
- esquerda: seta para esquerda
- direita: seta para direita
- botão A: 'x'
- botão B: 'z'
- select : 'a'
- start: 's'

A função UpdateControls verifica se as teclas seta para cima, seta para baixo, seta para esquerda, seta para direita, 'x', 'z', 'a' e 's' estão pressionadas ou não e armazena o resultado em variáveis globais. A função `ReadMemory` utiliza essas variáveis quando é feita uma leitura ao endereço do *gamepad* ($FF00).

6.9 Módulo Disassembler

O Módulo Disassembler implementa a seguinte caixa de diálogo:

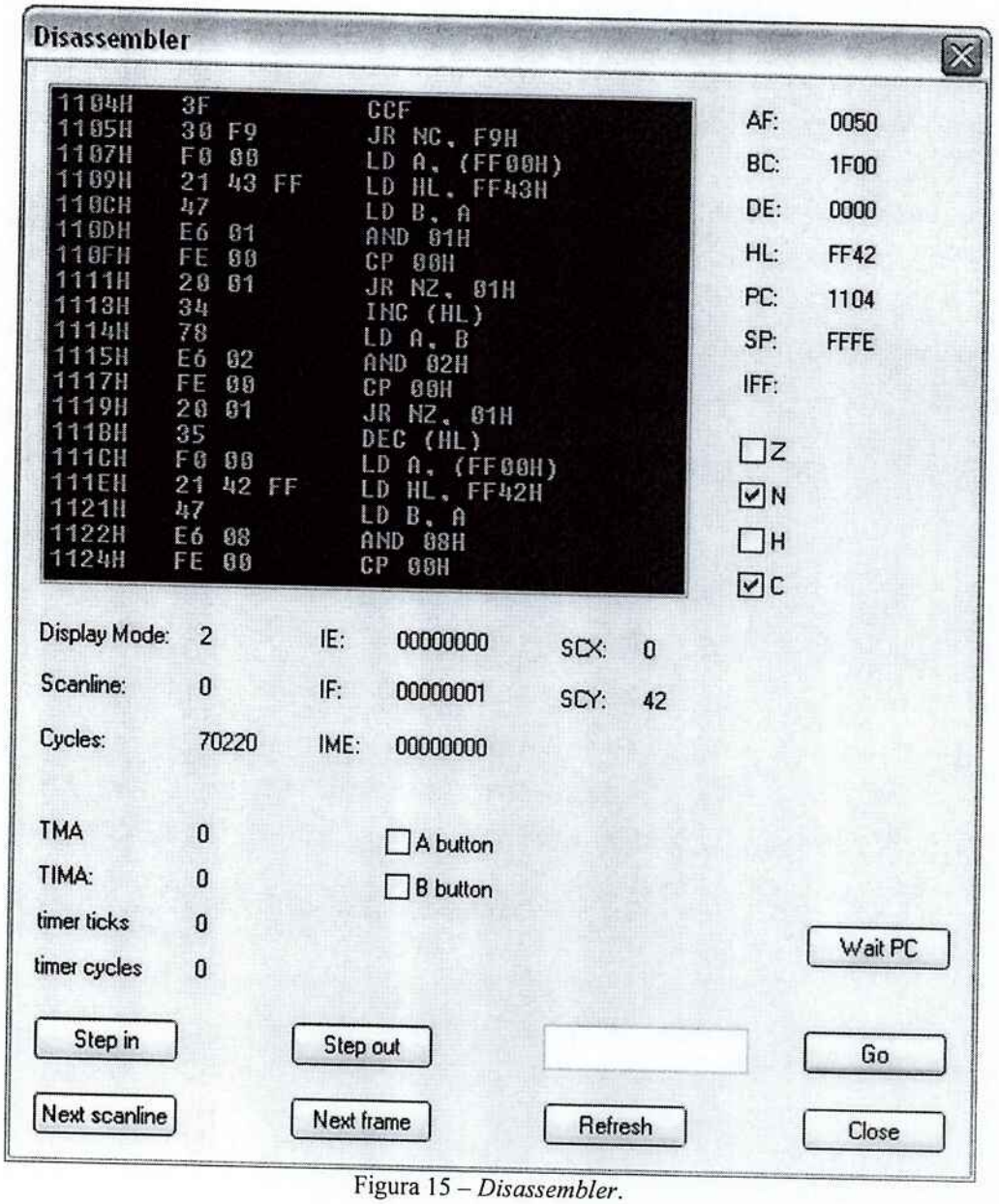

Figura 15 – *Disassembler*.

Trata-se de uma janela que mostra o pedaço do programa atualmente sendo executado pelo emulador. A janela possui também botões com funções de debug (depuração) de código Z80. Estas funcionalidades são de grande valia para desenvolvedores de software para Gameboy.

As funcionalidades são:

- *Step in*: executa uma instrução.
- *Step out*: executa infinitas instruções até que uma instrução RET seja alcançada.
- *Next scanline*: executa instruções até que a *scanline* corrente seja totalmente desenhada.
- *Next frame*: executa instruções até que o quadro corrente seja completamente desenhado.
- *Wait* PC: executa instruções até que a execução do programa atinja o endereço inserido pelo usuário na caixa de texto da janela.
- *Go*: executa instruções até que seja gasta a quantidade de ciclos de *clock* da CPU inserida pelo usuário na caixa de texto da janela

6.10 Módulo Header Viewer

O Módulo Header Viewer implementa a seguinte caixa de diálogo:

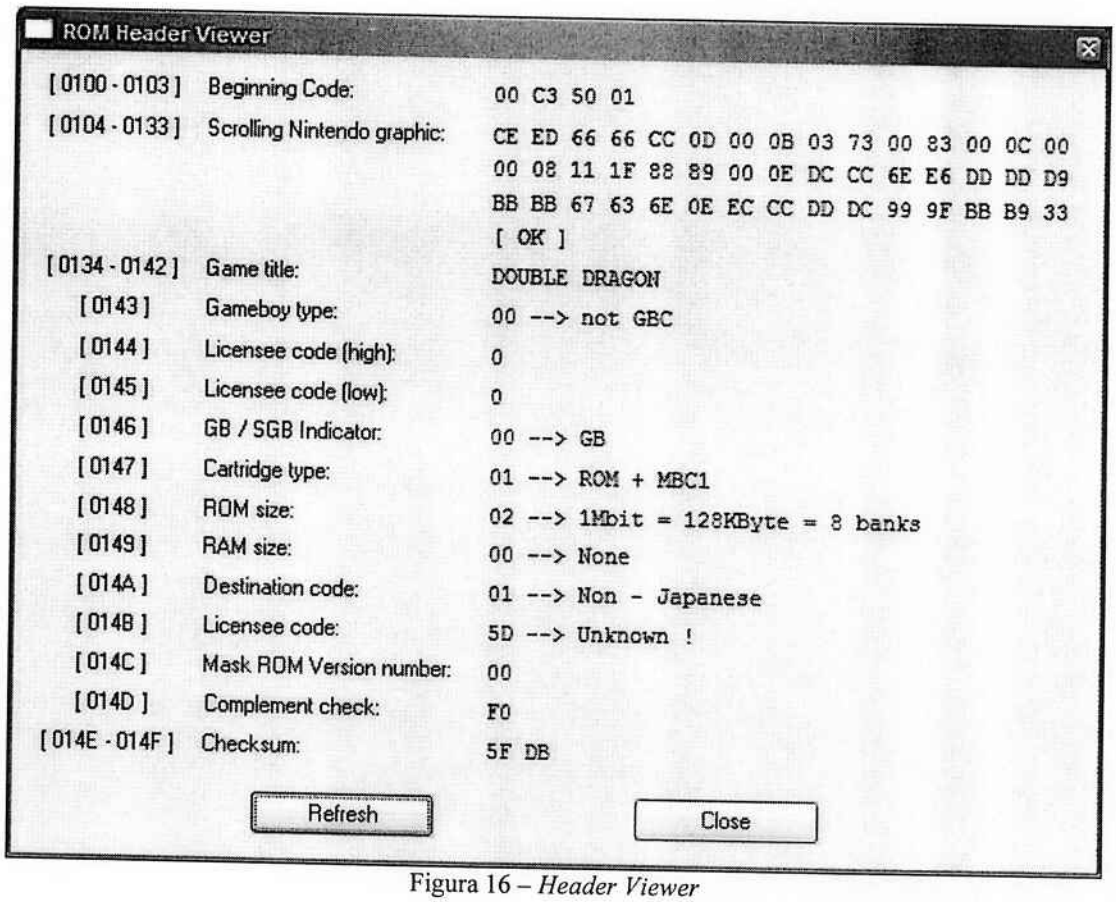

Figura 16 – *Header Viewer*

Essa janela exibe informações a respeito da imagem de cartucho carregada, que ficam disponíveis nos endereços $0100 a $014F.

6.11 Módulo Map Viewer

O Módulo Map Viewer implementa a seguinte caixa de diálogo:

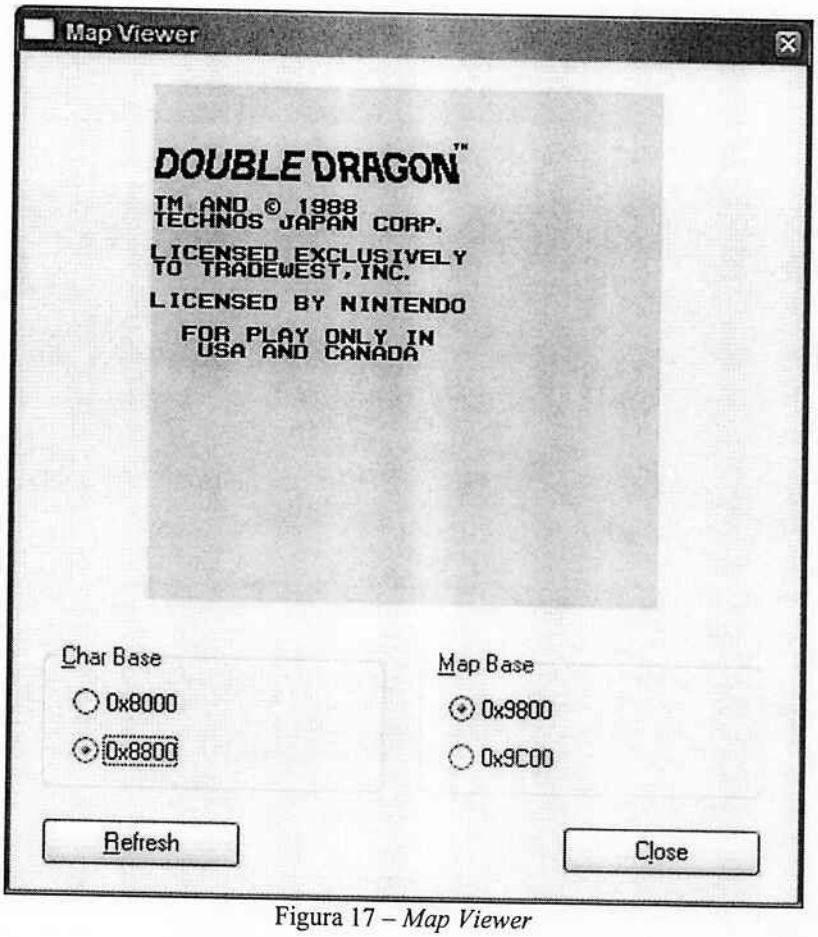

Figura 17 – *Map Viewer*

Essa janela exibe o conteúdo da parte da memória de vídeo que contem a camada de background e janela.

6.12 Módulo Memory Viewer

O Módulo Memory Viewer implementa a seguinte caixa de diálogo:

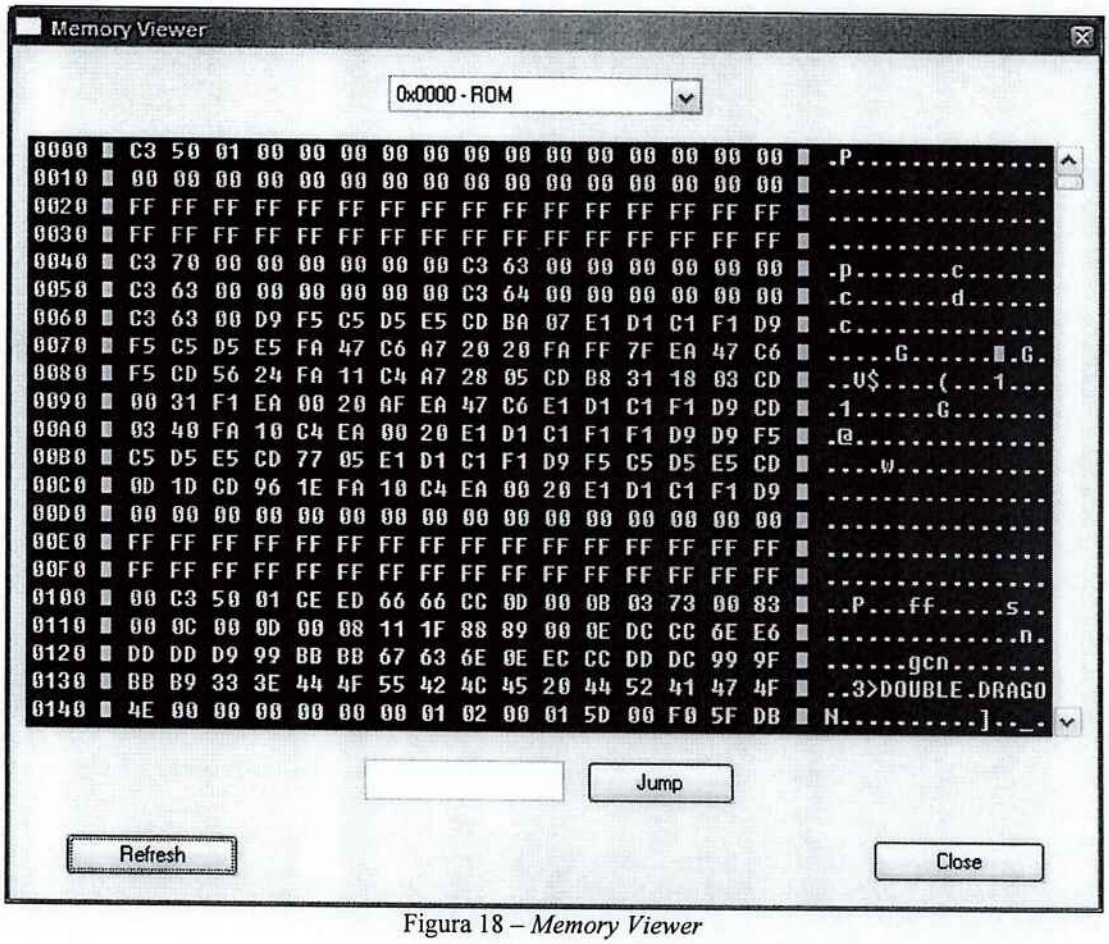

Figura 18 – *Memory Viewer*

Trata-se de uma janela que exibe o conteúdo atual de toda a memória.

6.13 Módulo OAM Viewer

O Módulo Memory Viewer implementa a seguinte caixa de diálogo:

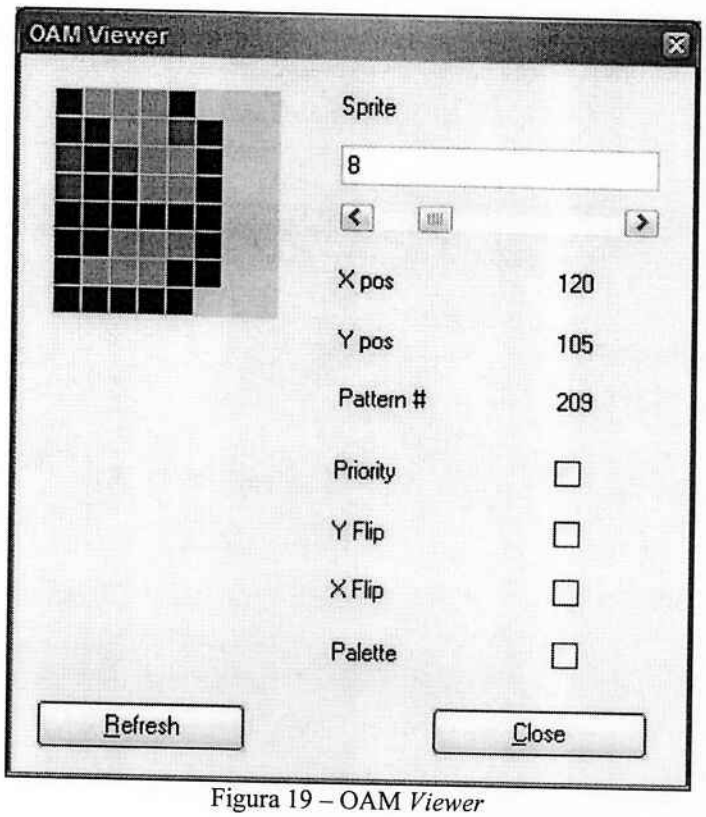

Figura 19 – OAM *Viewer*

Essa janela exibe o conteúdo da parte da memória de vídeo que contem a camada de *sprites* e os seus respectivos atributos.

6.14 Módulo Tile Viewer

O Módulo Tile Viewer implementa a seguinte caixa de diálogo:

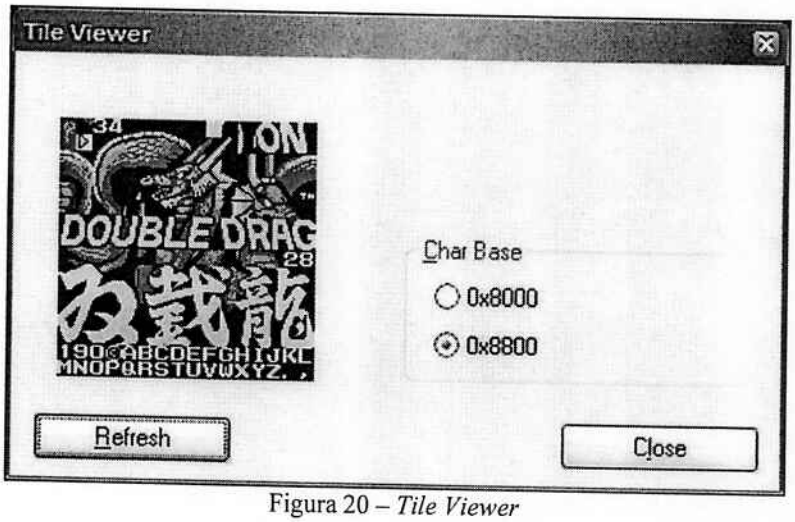

Figura 20 – *Tile Viewer*

Essa janela exibe todas as definições de caracteres atualmente na memória de vídeo.

7 TESTES E AVALIAÇÃO

Os testes efetuados para avaliação do emulador consistiram basicamente em rodar os jogos do GameBoy e GameBoy Color no emulador e comparar o funcionamento em ambos os sistemas (os jogos foram obtidos na Internet em *sites* especializados). Além dos jogos muitos programas de demonstração foram utilizados para testar diversas características do sistema emulado.

Os seguintes testes foram aplicados à versão final do emulador:

Teste	Resultado
Testar as funcionalidades do programa nos sistemas operacionais Windows 98 / NT / 2000 / XP / 2003.	Funcionalidade completa em todos os sistemas operacionais.
Testar a compatibilidade do emulador através da execução de mais de 200 ROM's de GB e GBC.	70% de compatibilidade.
Testar a robustez do programa ao tentar executar ROM's defeituosas ou arquivos que não são ROM's	O programa não trava.
Testar a performance do emulador através da execução do mesmo em máquinas antigas com processadores relativamente lentos.	O emulador roda com velocidade máxima (60 quadros por segundo) mesmo em sistemas antigos como 486s

Tabela 16 – Testes

Também conseguimos obter uma interface amigável para o emulador, característica esperada quando foram definidos os requisitos do projeto.

Com os dados obtidos nos testes a equipe chegou à conclusão de que o produto final do projeto satisfez todos os seus requisitos funcionais e não funcionais, estando de acordo com suas especificações.

8 CONSIDERAÇÕES FINAIS

8.1 Conquistas

A equipe acredita que após o longo caminho tomado pelo trabalho, todas as expectativas com relação ao aprendizado provindo do projeto foram alcançadas. Foram estudados vários aspectos ligados à emulação e arquitetura de computadores. O completo entendimento do hardware do GameBoy permitiu uma melhor visão de como diversos tipos de computadores funcionam. O estudo de lógicas de otimização ajudaram a melhoras as habilidades de programação dos membros da equipe, além de prover valiosas informações sobre o processo de compilação de linguagem de alto nível. Porém, a principal conquista está no fato de que a equipe pode aprender como os emuladores funcionam, como são projetados e construídos. Como citado no inicio do documento, esse aprendizado foi a principal razão para a escolha do projeto pela equipe.

8.2 Continuidade do projeto

Apesar de o projeto ter sido concluído com sucesso, o produto final do mesmo pode ser melhorado de várias maneiras. O aumento da compatibilidade do emulador, assim como a adição de novos filtros de imagem seriam os primeiros objetivos em uma possível continuação do projeto.

ANEXO A – LISTA DE REGISTRADORES DE E/S E SUAS FUNCIONALIDADES

FF00 – P1 (Leitura e Escrita)

Registrador para leitura de informações do controle do GameBoy e para determinação do tipo do sistema.

Bit 7 – não utilizado
Bit 6 – não utilizado
Bit 5 – P15 porta de saída
Bit 4 – P14 porta de saída
Bit 3 – P13 porta de entrada
Bit 2 – P12 porta de entrada
Bit 1 – P11 porta de entrada
Bit 0 – P10 porta de entrada

Escrevendo-se $20 no registrador, ativa-se P14 (*low-active*). Lendo então o registrador, os bits menos significativos vão representar, respectivamente, os botões: Direita, Esquerda, Cima e Baixo.
Escrevendo-se $10 no registrador, ativa-se P15 (*low-active*). Lendo então o registrador, os bits menos significativos vão representar, respectivamente, os botões: A, B, Select e Start.

FF01 – SB (Leitura e Escrita)

Dado para transferência serial.
1 byte de dado a ser lido ou escrito.

FF02 – SC (Leitura e Escrita)

Serial Control.

Bit 7 – *Flag* de início de transferência.

0: Sem transferência

1: Inicia transferência

Bit 0 – *Shift Clock*

0: Clock externo (500 KHz máximo)

1: Clock interno (8192 Hz)

A transferência é iniciada ativando o bit 7 do registrador. A transmissão e recepção serial são simultâneas. O dado recebido é automaticamente armazenado no registrador SB.

FF04 – DIV (Leitura e Escrita)

DIvider Register

Este registrador é incrementado 16384 vezes por segundo. Escrever qualquer valor nele o zera.

FF05 – TIMA (Leitura e Escrita)

Timer Counter

Este timer é incrementado pelo *clock* especificado pelo registrador TAC ($FF07). O *timer* gera interrupção quando sofre *overflow*.

FF06 – TMA (Leitura e Escrita)

Timer Modulo

Quando o TIMA sofre *overflow*, o valor deste registrador é carregado nele.

FF07 – TAC (Leitura e Escrita)

Timer Control

Bit 2

0: Pára o *timer*

1: Inicia o *timer*

Bits 1 + 0 – Seleção do clock

00: 4,096 KHz

01: 262,144 KHz

10: 65,536 KHz

11: 16,384 KHz

FF0F – IF (Leitura e Escrita)

Interrupt Flag

Bit 4 – Transição *high to low* ocorrida nas portas de entrada P10 - P13. Pulo para $0060

Bit 3 – Transferência serial completa. Pulo para $0058

Bit 2 – *Timer overflow*. Pulo para $0050.

Bit 1 – LCDC (ver registrador STAT). Pulo para $0048.

Bit 0 – Varredura vertical. Pulo para $0040.

A prioridade das interrupções é decrescente tal que a interrupção por varredura vertical tem a maior precedência dentre as interrupções.

FF10 – FF3F – Registradores do PSG

A descrição dos mesmo está no documento GBSOUND [6].

FF40 – LCDC (Leitura e Escrita)

LCD *Control*

Bit 7 – Controle da operação do LCD

0: Desativar LCD.

1: Ativar LCD.

Bit 6 – Seleção do mapa de caracteres da janela

0: $9800-$9BFF

1: $9C00-$9FFF

Bit 5 – Exibição da janela

0: Desligado.

1: Ativado.

Bit 4 – Seleção do TDT para o *background* e janela

0: $8800-$97FF

1: $8000-$8FFF

Bit 3 – Seleção do mapa de caracteres do *background*

0: $9800-$9BFF

1: $9C00-$9FFF

Bit 2 – Tamanho do *Sprite*

0: 8x8.

1: 8x16. (largura x comprimento).

Bit 1 – Exibição dos *sprites*

0: Desligado.

1: Ativado.

Bit 0 – Exibição do background e janela

0: Desligado.

1: Ativado.

FF41 – STAT (Leitura e Escrita)

LCDC Status

Bits 6-3 – Seleção de interrupções do LCD ativas

Bit 6 – *Flag* de coincidência.

Bit 5 – Modo 10

Bit 4 – Modo 01

Bit 3 – Modo 00

0: Desativado

1: Ativado

Bit 2 – *Flag* de coincidência.

0: LYC diferente de LY.

1: LYC igual ao LY.

Bit 1-0 – *Mode Flag*

00: Varredura horizontal (CPU pode acessar a VRAM).

01: Varredura vertical (CPU pode acessar a VRAM).

10: Acesso à OAM.

11: Acesso à OAM e à VRAM.

FF42 – SCY (Leitura e Escrita)

Scroll Y

Byte com valor do scroll Y do *background.*

FF43 – SCX (Leitura e Escrita)

Scroll X

Byte com valor do scroll X do *background.*

FF44 – LY (Leitura apenas)

LCDC coordenada Y

Este registrador indica a *scanline* que está sendo desenhada no momento. Valores de 144 a 153 indicam varredura vertical.

FF45 – LYC (Leitura e Escrita)

LY *Compare*

Este registrador se compara com o LY, caso os valores sejam iguais ele causa a ativação do *flag* de coincidência do registrador STAT.

FF46 – DMA (Escrita apenas)

No GB pode-se efetuar uma transferência por DMA da ROM ou RAM para a OAM ($FE00-$FE9F). A transferência dos 40*28 bits demora 160ms para ser concluída. Para executar a transferência basta escrever o endereço inicial dos dados neste registrador.

FF47 – BGP (Leitura e Escrita)

Palheta para o *background* e janela.

Bit 7-6 – Nível de cinza para a cor 11 (normalmente a cor mais escura)
Bit 5-4 – Nível de cinza para a cor 10
Bit 3-2 – Nível de cinza para a cor 01
Bit 1-0 – Nível de cinza para a cor 00 (normalmente a cor mais clara)

FF48 – OBP0 (Leitura e Escrita)

Object Palette Data 0

Primeira palheta (0) para os *sprites*

Funciona do mesmo modo que o registrador BGP ($FF47), porém a cor 0 é transparente.

FF49 – OBP1 (Leitura e Escrita)

Object Palette Data 1

Primeira palheta (1) para os *sprites*

Funciona do mesmo modo que o registrador BGP ($FF47), porém a cor 0 é transparente.

FF4A – WY (Leitura e Escrita)

Posição Y da janela no *background.*
WY deve estar entre 0 e 143 inclusos.

FF4B – WX (Leitura e Escrita)

Posição X da janela no *background.*
WX deve estar entre 0 e 166 inclusos.

FFFF – IE (Leitura e Escrita)

Interrupt Enable

Bit 4 – Transição *high to low* ocorrida nas portas de entrada P10 - P13. Pulo para $0060

Bit 3 – Transferência serial completa. Pulo para $0058

Bit 2 – *Timer overflow*. Pulo para $0050.

Bit 1 – LCDC (ver registrador STAT). Pulo para $0048.

Bit 0 – Varredura vertical. Pulo para $0040.

0: Desabilita interrupção.

1: Habilita interrupção.

ANEXO B – LISTA DE INSTRUÇÕES DO PROCESSADOR DO GAMEBOY

As listas a seguir apresentam as instruções do processador do GameBoy, seus respectivos *opcodes* e o número de ciclos que cada instrução toma.

8bit Load Commands

```
  ld   r,r          xx          4 ---- r=r
  ld   r,n          xx nn       8 ---- r=n
  ld   r,(HL)       xx          8 ---- r=(HL)
  ld   (HL),r       7x          8 ---- (HL)=r
  ld   (HL),n       36 nn      12 ----
  ld   A,(BC)       0A          8 ----
  ld   A,(DE)       1A          8 ----
  ld   A,(nn)       FA         16 ----
  ld   (BC),A       02          8 ----
  ld   (DE),A       12          8 ----
  ld   (nn),A       EA         16 ----
  ld   A,(FF00+n)   F0 nn      12 ---- read from io-port n (memory
FF00+n)
  ld   (FF00+n),A   E0 nn      12 ---- write to io-port n (memory
FF00+n)
  ld   A,(FF00+C)   F2          8 ---- read from io-port C (memory
FF00+C)
  ld   (FF00+C),A   E2          8 ---- write to io-port C (memory
FF00+C)
  ldi  (HL),A       22          8 ---- (HL)=A, HL=HL+1
  ldi  A,(HL)       2A          8 ---- A=(HL), HL=HL+1
  ldd  (HL),A       32          8 ---- (HL)=A, HL=HL-1
  ldd  A,(HL)       3A          8 ---- A=(HL), HL=HL-1
```

16bit Load Commands

```
  ld   rr,nn        x1 nn nn   12 ---- rr=nn (rr may be BC,DE,HL or
SP)
  ld   SP,HL        F9          8 ---- SP=HL
  push rr           x5         16 ---- SP=SP-2  (SP)=rr   (rr may be
BC,DE,HL,AF)
  pop  rr           x1         12 (AF) rr=(SP)  SP=SP+2   (rr may be
BC,DE,HL,AF)
```

8bit Arithmetic/Logical Commands

```
  add  A,r          8x          4 z0hc A=A+r
  add  A,n          C6 nn       8 z0hc A=A+n
  add  A,(HL)       86          8 z0hc A=A+(HL)
  adc  A,r          8x          4 z0hc A=A+r+cy
  adc  A,n          CE nn       8 z0hc A=A+n+cy
  adc  A,(HL)       8E          8 z0hc A=A+(HL)+cy
  sub  r            9x          4 z1hc A=A-r
  sub  n            D6 nn       8 z1hc A=A-n
  sub  (HL)         96          8 z1hc A=A-(HL)
  sbc  A,r          9x          4 z1hc A=A-r-cy
  sbc  A,n          DE nn       8 z1hc A=A-n-cy
  sbc  A,(HL)       9E          8 z1hc A=A-(HL)-cy
  and  r            Ax          4 z010 A=A & r
```

```
and   n          E6 nn      8 z010 A=A & n
and   (HL)       A6         8 z010 A=A & (HL)
xor   r          Ax         4 z000
xor   n          EE nn      8 z000
xor   (HL)       AE         8 z000
or    r          Bx         4 z000 A=A | r
or    n          F6 nn      8 z000 A=A | n
or    (HL)       B6         8 z000 A=A | (HL)
cp    r          Bx         4 z1hc compare A-r
cp    n          FE nn      8 z1hc compare A-n
cp    (HL)       BE         8 z1hc compare A-(HL)
inc   r          xx         4 z0h- r=r+1
inc   (HL)       34        12 z0h- (HL)=(HL)+1
dec   r          xx         4 z1h- r=r-1
dec   (HL)       35        12 z1h- (HL)=(HL)-1
daa              27         4 z-0x decimal adjust akku
cpl              2F         4 -11- A = A xor FF
```

16bit Arithmetic/Logical Commands

```
  add  HL,rr     x9          8 -0hc HL = HL+rr     ;rr may be
BC,DE,HL,SP
  inc  rr        x3          8 ---- rr = rr+1      ;rr may be
BC,DE,HL,SP
  dec  rr        xB          8 ---- rr = rr-1      ;rr may be
BC,DE,HL,SP
  add  SP,dd     E8         16 00hc SP = SP +/- dd ;dd is 8bit
signed number
  ld   HL,SP+dd  F8         12 00hc HL = SP +/- dd ;dd is 8bit
signed number
```

Rotate and Shift Commands

```
rlca           07          4 000c rotate akku left
rla            17          4 000c rotate akku left through carry
rrca           0F          4 000c rotate akku right
rra            1F          4 000c rotate akku right through carry
rlc   r        CB 0x       8 z00c rotate left
rlc   (HL)     CB 06      16 z00c rotate left
rl    r        CB 1x       8 z00c rotate left through carry
rl    (HL)     CB 16      16 z00c rotate left through carry
rrc   r        CB 0x       8 z00c rotate right
rrc   (HL)     CB 0E      16 z00c rotate right
rr    r        CB 1x       8 z00c rotate right through carry
rr    (HL)     CB 1E      16 z00c rotate right through carry
sla   r        CB 2x       8 z00c shift left arithmetic (b0=0)
sla   (HL)     CB 26      16 z00c shift left arithmetic (b0=0)
swap  r        CB 3x       8 z000 exchange low/hi-nibble
swap  (HL)     CB 36      16 z000 exchange low/hi-nibble
sra   r        CB 2x       8 z00c shift right arithmetic (b7=b7)
sra   (HL)     CB 2E      16 z00c shift right arithmetic (b7=b7)
srl   r        CB 3x       8 z00c shift right logical (b7=0)
srl   (HL)     CB 3E      16 z00c shift right logical (b7=0)
```

Singlebit Operation Commands

```
bit   n,r      CB xx       8 z01- test bit n
bit   n,(HL)   CB xx      12 z01- test bit n
set   n,r      CB xx       8 ---- set bit n
```

```
  set  n,(HL)     CB xx        16 ---- set bit n
  res  n,r        CB xx         8 ---- reset bit n
  res  n,(HL)     CB xx        16 ---- reset bit n
```

CPU Control Commands

```
  ccf             3F            4 -00c cy=cy xor 1
  scf             37            4 -001 cy=1
  nop             00            4 ---- no operation
  halt            76          N*4 ---- halt until interrupt occurs
(low power)
  stop            10 00         ? ---- low power standby mode (VERY
low power)
  di              F3            4 ---- disable interrupts, IME=0
  ei              FB            4 ---- enable interrupts, IME=1
```

Jump Commands

```
  jp   nn         C3 nn nn     16 ---- jump to nn, PC=nn
  jp   HL         E9            4 ---- jump to HL, PC=HL
  jp   f,nn       xx nn nn 16;12 ---- conditional jump if nz,z,nc,c
  jr   PC+dd      18 dd        12 ---- relative jump to nn (PC=PC+/-
7bit)
  jr   f,PC+dd    xx dd      12;8 ---- conditional relative jump if
nz,z,nc,c
  call nn         CD nn nn     24 ---- call to nn, SP=SP-2, (SP)=PC,
PC=nn
  call f,nn       xx nn nn 24;12 ---- conditional call if nz,z,nc,c
  ret             C9           16 ---- return, PC=(SP), SP=SP+2
  ret  f          xx         20;8 ---- conditional return if nz,z,nc,c
  reti            D9           16 ---- return and enable interrupts
(IME=1)
  rst  n          xx           16 ---- call to 00,08,10,18,20,28,30,38
```

ANEXO C – CRONOGRAMA FINAL DO PROJETO

Task Name	Duration	Start	Finish	Predecessors	Resource Names
Estudo da arquitetura do Gameboy	15 days	Mon 12/4/04	Fri 30/4/04		Adriano, Eduardo
Estabelecimento do ambiente de desenvolvimento	10 days	Mon 3/5/04	Fri 14/5/04	1	Adriano, Eduardo
Desenvolvimento do programa de demonstração para prototipação.	5 days	Mon 17/5/04	Fri 21/5/04	2	Adriano, Eduardo
Desenvolvimento do protótipo	20 days	Mon 24/5/04	Fri 18/6/04	3	Adriano, Eduardo
Testes do protótipo	10 days	Mon 21/6/04	Fri 2/7/04	4	Adriano, Eduardo
Desenvolvimento do Núcleo da CPU	20 days	Mon 5/7/04	Fri 30/7/04	5	Adriano
Atualização da documentação	20 days	Mon 5/7/04	Fri 30/7/04		Eduardo
Desenvolvimento do VDP	20 days?	Mon 2/8/04	Fri 27/8/04	7	Adriano
Desenvolvimento do PSG	30 days	Mon 2/8/04	Fri 10/9/04	6	Eduardo
Desenvolvimento do dispositivo de controle (gamepad)	10 days	Mon 30/8/04	Fri 10/9/04	9	Adriano
Finalização do desenvolvimento do PSG	10 days	Mon 13/9/04	Fri 24/9/04	8	Eduardo
Testes e Ajustes	10 days	Mon 13/9/04	Fri 24/9/04	11	Adriano, Eduardo
Desenvolvmiento de Funcionalidades Adicionais	20 days	Mon 27/9/04	Fri 22/10/04	12	Adriano, Eduardo
Documentação	30 days	Mon 25/10/04	Fri 3/12/04	13	Adriano, Eduardo
Testes	30 days	Mon 25/10/04	Fri 3/12/04		Adriano, Eduardo

LISTA DE REFERÊNCIAS

[1] ZiLOG Worldwide Headquarters. **Z80 Family CPU User Manual.** Campbell: 2001. 306 p.

[2] Intel Corporation. **Intel. IA-32 Intel Architecture Optimization Reference Manual.** 2004.

[3] Intel Corporation. **Intel. IA-32 Intel Architecture Developer's Manual Volume 1: Basic Architecture.** 2004.

[4] Intel Corporation. **Intel. IA-32 Intel Architecture Developer's Manual Volume 2A/2B: Instruction Set Reference.** 2004.

[5] Frohwein. J. EUA. **Apresenta informações sobre desenvolvimento de software para alguns videogames**. Disponível em: <http://www.devrs.com>. Acesso em: 03 de dezembro de 2004.

[6] Belogic Software. **Apresenta informações sobre o PSG do GBA, que é o mesmo do GB e GBC.** Disponível em: <http://belogic.com/gba>. Acesso em: 03 de dezembro de 2004.

APÊNDICE I – CD COM ARQUIVOS DO PROJETO

O CD do projeto, PCS2050-01-CD-2004, contém os seguintes diretórios e arquivos:

/readme.txt

Arquivo contendo informações sobre o CD.

/docs/

Diretório contendo toda a documentação do projeto, incluído o documento final.

/src

Diretório contendo os arquivos fonte do emulador.

/bin

Diretório contendo o executável da última versão do emulador.

/roms

Diretório contendo algumas ROM's de jogos de GameBoy para serem testadas no emulador.