INSTITUTO FEDERAL DE EDUCAÇÃO, CIÊNCIA E TECNOLOGIA DO PIAUÍ
CAMPUS TERESINA CENTRAL
CURSO TECNOLOGIA EM ANÁLISE E DESENVOLVIMENTO DE SISTEMAS

FRANCISCO ALEXANDRO CONCEIÇÃO SOUSA

DESENVOLVIMENTO DE UM EMULADOR CHIP-8 COMO ESTRATÉGIA PARA O ENSINO DE ARQUITETURA DE COMPUTADORES

TERESINA

2025

FRANCISCO ALEXANDRO CONCEIÇÃO SOUSA

DESENVOLVIMENTO DE UM EMULADOR CHIP-8 COMO ESTRATÉGIA PARA O ENSINO DE ARQUITETURA DE COMPUTADORES

Trabalho de Conclusão de Curso (monografia) apresentado como exigência parcial para obtenção do diploma do Curso de Tecnologia em Análise e Desenvolvimento de Sistemas do Instituto Federal de Educação, Ciência e Tecnologia do Piauí, Campus Teresina Central.

Orientador: Prof. Me. Francisco Marcelino Almeida de Araújo.

TERESINA

2025

Dados Internacionais de Catalogação na Publicação (CIP) de acordo com ISBD

Sousa, Francisco Alexandro Conceição

S725d Desenvolvimento de um emulador CHIP-8 como estratégia para o ensino de arquitetura de computadores / Francisco Alexandro Conceição Sousa. - 2025.

48 f.: il.

Trabalho de Conclusão de Curso (Tecnologia em Análise e Desenvolvimento de Sistemas) - Instituto Federal de Educação, Ciência e Tecnologia do Piauí, Campus Teresina Central, 2025.

Orientador : Prof Me. Francisco Marcelino Almeida Araújo.

1. Emulação . 2. Educação. 3. CHIP-8. 4. Computação. 5. Preservação digital. I. Título.

CDD - 004.21

Elaborado por Tanize Maria Sales CRB 3/1024

FRANCISCO ALEXANDRO CONCEIÇÃO SOUSA

DESENVOLVIMENTO DE UM EMULADOR CHIP-8 COMO ESTRATÉGIA PARA O ENSINO DE ARQUITETURA DE COMPUTADORES

Trabalho de Conclusão de Curso (monografia) apresentado como exigência parcial para obtenção do diploma do Curso de Tecnologia em Análise e Desenvolvimento de Sistemas do Instituto Federal de Educação, Ciência e Tecnologia do Piauí, Campus Teresina Central .

Aprovada em: 16/12/2025.

BANCA EXAMINADORA

_____
Prof. Me. Francisco Marcelino Almeida de Araújo (Orientador)
Instituto Federal do Piauí (IFPI)

_____
Prof. Me. Fernando Castelo Branco Gonçalves Santana
Instituto Federal do Piauí (IFPI)

_____
Prof. Me. Wilson de Oliveira Junior
Instituto Federal do Piauí (IFPI)

_____
Antonio Kauê Cavalcante Ferreira
Instituto Federal do Piauí (IFPI)

Dedico este trabalho ao meu eu do futuro, com a esperança de que, ao revisitar estas páginas, reconheça que toda a dedicação e persistência valeram a pena.

AGRADECIMENTOS

Aos meus educadores, por despertarem em mim o interesse pelo conhecimento e por me guiarem na construção das bases que tornaram possível a realização deste trabalho.

Aos meus pais, que, mesmo diante das dificuldades, nunca deixaram de me apoiar e de investir na minha educação e formação, sendo meu maior exemplo de dedicação e perseverança.

À minha namorada, pelo apoio, compreensão e incentivo incondicionais ao longo desta trajetória, e por me inspirar diariamente a buscar ser uma pessoa melhor.

Aos meus amigos, pela amizade constante, pelas palavras de apoio e pelas conversas que aliviaram os momentos de maior tensão — em especial ao meu amigo Gabriel, pela parceria e pela boa energia durante todo o percurso.

RESUMO

Este trabalho apresenta o desenvolvimento de um emulador do sistema CHIP-8, proposto como uma ferramenta didática voltada ao ensino e à compreensão de conceitos fundamentais de Arquitetura de Computadores. A iniciativa busca unir aprendizado prático e preservação histórica, permitindo que estudantes e entusiastas explorem, de forma interativa, o funcionamento de uma arquitetura clássica e simplificada da computação. O projeto foi concebido com o propósito de facilitar o entendimento dos princípios que regem o funcionamento interno de uma arquitetura computacional, abordando temas como execução de instruções, uso de registradores, manipulação de memória e controle de fluxo. Além disso, demonstra como tecnologias modernas podem ser aplicadas à reconstrução e modernização de sistemas retro, oferecendo uma abordagem acessível, visual e intuitiva. Dessa forma, o emulador contribui para o fortalecimento do aprendizado prático e para o resgate de arquiteturas históricas, promovendo a integração entre teoria, prática e inovação tecnológica.

Palavras-chave: emulação; CHIP-8; educação; computação; preservação digital.

ABSTRACT

This work presents the development of a CHIP-8 system emulator, proposed as a didactic tool aimed at teaching and understanding fundamental concepts of Computer Architecture. The initiative seeks to combine practical learning and historical preservation, allowing students and enthusiasts to interactively explore the operation of a classical and simplified computing architecture. The project was conceived with the purpose of facilitating the understanding of the principles that govern the internal functioning of a computational architecture, addressing topics such as instruction execution, register usage, memory manipulation, and flow control. Furthermore, it demonstrates how modern technologies can be applied to the reconstruction and modernization of retro systems, offering an accessible, visual, and intuitive approach. In this way, the emulator contributes to strengthening practical learning and rescuing historical architectures, promoting the integration of theory, practice, and technological innovation.

Keywords: emulation; CHIP-8; computer architecture; educational technology; digital preservation.

LISTA DE ILUSTRAÇÕES

LISTA DE ABREVIATURAS E SIGLAS

ALU	*Arithmetic Logic Unit* (Unidade Lógica e Aritmética)
CDP-1802	Microprocessador utilizado no sistema COSMAC VIP
CHIP-8	*Comprehensive Hexadecimal Interpretive Programming 8-Bit*
CPU	*Central Processing Unit* (Unidade Central de Processamento)
DT	*Delay Timer* (Temporizador de Atraso)
FPGAs	*Field-Programmable Gate Arrays*
HLE	*High-Level Emulation* (Emulação de Alto Nível)
I	Registrador de Índice
LLE	*Low-Level Emulation* (Emulação de Baixo Nível)
MMU	*Memory Management Unit* (Unidade de Gerenciamento de Memória)
PC	*Program Counter* (Contador de Programa)
RAM	*Random Access Memory* (Memória de Acesso Aleatório)
ROM	*Read-Only Memory* (Memória Somente de Leitura)
SCHIP	SUPER-CHIP (Variante do CHIP-8)
SP	*Stack Pointer* (Ponteiro de Pilha)
ST	*Sound Timer* (Temporizador de Som)
UI	Interface do Usuário
VF	Registrador de Status (*Flag*)
VIP	*Virtual Interpreter Program* (presente no nome COSMAC VIP)
Vx/Vy	Registradores de uso geral do CHIP-8
XOR	*Exclusive OR Logic* (Operação lógica "ou exclusivo" usada na renderização)
XO-CHIP	Variante moderna do CHIP-8

x86/ARM Arquiteturas modernas de processadores

SUMÁRIO

1 INTRODUÇÃO

Em 1837, Charles Babbage idealizou uma máquina que poderia realizar cálculos de forma autônoma — a Máquina Analítica. Décadas depois, Alan Turing definiria matematicamente o conceito de computação universal, enquanto John von Neumann consolidaria os princípios que, ainda hoje, orientam a estrutura dos computadores modernos. Desde então, a humanidade não apenas se dedica a criar novas tecnologias, mas também a compreender e reviver aquelas que marcaram o início dessa trajetória.

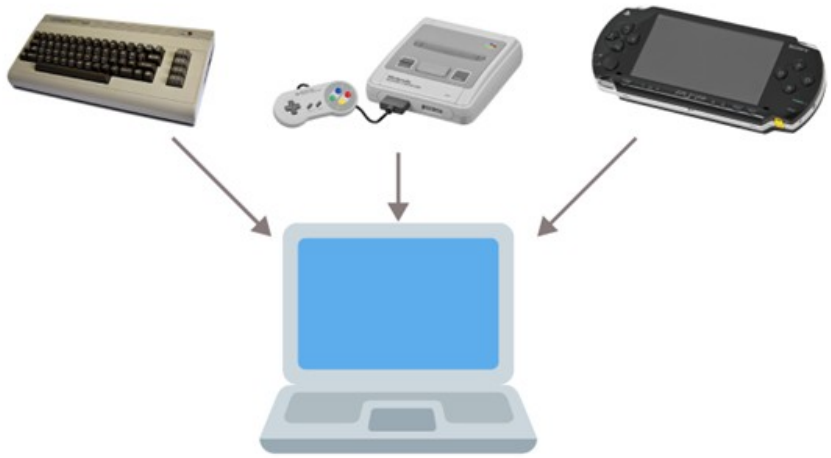
Essa busca por reviver o passado tecnológico revela um aspecto fundamental do espírito científico: compreender as bases sobre as quais o presente foi construído. Por mais que os sistemas modernos avancem em complexidade e desempenho, é no estudo das origens — das máquinas e das ideias que as sustentaram — que se encontra a verdadeira compreensão do funcionamento computacional.

É nesse contexto que surge a emulação, um processo que possibilita trazer de volta à vida sistemas e softwares de outras épocas, permitindo que sejam executados em plataformas contemporâneas. Conforme definido por Smith e Nair, a emulação envolve a reprodução do comportamento de uma arquitetura de conjunto de instruções (ISA) alvo em um host com uma ISA diferente, utilizando camadas de *software* para a tradução de instruções (SMITH; NAIR, 2005). Em termos simples, emular significa reproduzir digitalmente um sistema de hardware para que programas criados para ele possam ser executados em outra plataforma.

A relevância dessa prática vai muito além do aspecto nostálgico, configurando-se como uma ferramenta essencial tanto para o estudo das arquiteturas computacionais quanto para a preservação dos marcos históricos da tecnologia. Para compreender em profundidade o processo de emulação, é necessário revisitar a Arquitetura de Von Neumann, proposta na década de 1940, em um contexto em que os computadores ainda eram sistemas experimentais e inflexíveis. Na época, máquinas como o ENIAC exigiam reconfiguração manual para cada tarefa, o que motivou John von Neumann a propor o conceito de programa armazenado — permitindo que instruções e dados compartilhassem a mesma memória. Essa ideia definiu a base da computação moderna, organizando o sistema em CPU, memória e dispositivos de entrada e saída interligados (NEUMANN, 1945).

Emular um sistema, portanto, consiste em reproduzir com fidelidade essas interações, buscando recriar o comportamento interno do hardware original em um ambiente distinto — o que representa o principal desafio técnico da emulação. A Figura 1 apresenta uma visão conceitual desse processo, no qual sistemas do passado são reinterpretados e executados em plataformas modernas.

Figura 1 – Representação conceitual da emulação de sistemas antigos em um computador moderno



Fonte: TEAM (2023).

O objeto de estudo deste trabalho é o CHIP-8, um sistema de interpretação desenvolvido na década de 1970 com o propósito de facilitar a criação de jogos e aplicações simples. Ao implementar um emulador desse sistema, busca-se não apenas reproduzir seu funcionamento, mas também investigar em profundidade suas características arquiteturais.

O foco deste projeto está na emulação por *software*, uma abordagem amplamente adotada que utiliza programas para simular o comportamento do hardware alvo. Embora existam alternativas baseadas em hardware — como o uso de FPGAs (*Field-Programmable Gate Arrays*) —, este estudo concentra-se exclusivamente na implementação e análise de um emulador puramente em *software*. Essa escolha se justifica por oferecer maior acessibilidade, menor custo e facilidade de experimentação, tornando o processo mais viável tanto para fins acadêmicos quanto para a pesquisa exploratória (IFURY25, 2021).

Considerando o potencial educacional e histórico do tema, a próxima seção

apresenta a justificativa deste trabalho, destacando sua relevância técnica e seu papel na preservação e compreensão dos fundamentos da computação.

1.1 JUSTIFICATIVA

Este trabalho destaca-se por sua relevância no ensino de conceitos fundamentais de emulação e arquitetura de computadores. A construção de um emulador do CHIP-8 proporciona um ambiente prático para a exploração desses temas, permitindo que conceitos teóricos sejam aplicados de forma concreta por meio de experimentos e implementações reais. Dessa forma, o projeto torna-se uma ferramenta didática valiosa, facilitando o entendimento de aspectos essenciais, como a execução de instruções e o funcionamento do *hardware*.

Além do aspecto educacional, este trabalho também contribui para a preservação da história da computação. O CHIP-8, apesar de sua simplicidade, possui grande valor histórico, representando uma etapa importante na evolução dos sistemas computacionais. Desenvolver um emulador é, portanto, uma forma de resgatar e preservar a memória desse período, ressaltando sua importância para a formação de novas gerações de programadores e pesquisadores.

Complementando esses aspectos, o projeto se destaca pela elaboração de documentação detalhada e materiais educacionais acessíveis, que funcionam como um guia abrangente para implementação e permitem que desenvolvedores e estudantes se aprofundem nos princípios de emulação e arquitetura de computadores. Por meio de explicações claras, exemplos práticos e reflexões obtidas durante o desenvolvimento, o trabalho busca tornar conceitos complexos mais compreensíveis e aplicáveis, transformando o aprendizado sobre a emulação do CHIP-8 em uma experiência envolvente, formativa e relevante para o avanço do conhecimento técnico.

A preservação digital de sistemas antigos é outro ponto relevante do projeto, sendo essencial para compreender a trajetória da computação e a evolução das tecnologias contemporâneas. A emulação constitui uma ferramenta indispensável nesse contexto. Conforme argumenta Rothenberg (1995), ela se estabelece como a única estratégia capaz de preservar o comportamento original de documentos e sistemas digitais a longo prazo, recriando o ambiente de hardware obsoleto em plataformas modernas.

Essa necessidade de independência do hardware original é ilustrada por Frank Cifaldi, fundador da *Video Game History Foundation.* Ele observa que, se filmes tivessem sido lançados apenas em fitas VHS, hoje seria necessário procurar cópias antigas e equipamentos funcionais sujeitos a falhas (CIFALDI, 2016); de forma análoga, muitos jogos e sistemas antigos correm o risco de desaparecer por existirem apenas em seus formatos físicos originais. Dessa maneira, a contribuição deste projeto para a preservação histórica representa um passo significativo na manutenção da memória dos sistemas computacionais, fornecendo uma base sólida para estudos futuros.

Por fim, o objetivo central deste projeto é apresentar os desafios e soluções técnicas de maneira clara e acessível, incentivando tanto especialistas quanto iniciantes a compreender a arquitetura de computadores. A emulação se mostra uma ferramenta poderosa nesse contexto, permitindo que conceitos teóricos, muitas vezes abstratos, sejam explorados de forma prática e interativa, como a execução de instruções, a gestão de memória e a interação entre diferentes componentes de *hardware* e *software*.

Essa abordagem torna o aprendizado mais concreto, facilitando a assimilação dos fundamentos da computação e promovendo uma compreensão mais profunda da lógica subjacente ao funcionamento das máquinas digitais. Além disso, a simplicidade relativa do sistema escolhido torna o estudo mais convidativo, permitindo que conceitos complexos sejam abordados de forma gradual e compreensível, favorecendo a consolidação do conhecimento.

1.2 OBJETIVO GERAL

O projeto tem como objetivo desenvolver uma ferramenta didática baseada na arquitetura CHIP-8, combinando fidelidade histórica e recursos modernos para facilitar a compreensão dos fundamentos da computação e promover aprendizado prático e acessível.

1.3 OBJETIVOS ESPECÍFICOS

- Implementar um emulador funcional do CHIP-8, capaz de interpretar e executar o conjunto completo de instruções.

- Replicar os principais componentes da arquitetura, incluindo CPU, registradores, memória, temporizadores e display gráfico.
- Desenvolver uma interface intuitiva e amigável, acessível a usuários com diferentes níveis de conhecimento técnico.
- Criar documentação detalhada que explique conceitos de emulação e arquitetura de computadores, servindo como guia para desenvolvedores e estudantes.

2 FUNDAMENTAÇÃO TEÓRICA

2.1 FUNDAMENTOS DA EMULAÇÃO

No campo da emulação, existem conceitos e terminologias essenciais para compreender como sistemas antigos são reproduzidos em plataformas modernas. Embora alguns desses termos possam não ser familiares em outros contextos computacionais, eles são fundamentais para a construção e operação de emuladores (SPASOJEVIC, 2024).

- **Target:** Refere-se ao sistema original que se deseja emular. No caso do CHIP-8, o target corresponde ao computador ou plataforma onde o código foi inicialmente executado, como o COSMAC VIP.
- **Host:** É o sistema onde o emulador será executado. Normalmente, trata-se de um computador moderno, com recursos superiores aos do target, como maior capacidade de processamento e memória, oferecendo o ambiente necessário para que o código do emulador simule corretamente o target.
- **ROM:** Refere-se ao *software* criado para ser executado no target. Em sistemas antigos, a ROM correspondia fisicamente a um chip de memória, como os cartuchos de jogos. Na emulação, a ROM é um arquivo digital que contém o código original, podendo ser um jogo ou uma aplicação desenvolvida para o target.

Um componente central para o funcionamento de qualquer emulador é a CPU, frequentemente chamada de “coração” de um sistema computacional (LENOVO,

2023). Enquanto a ROM armazena o *software* a ser executado, é a CPU que interpreta essas instruções e garante que o comportamento do programa seja reproduzido fielmente no ambiente do emulador. Para entender melhor esse processo, é necessário conhecer o ciclo *fetch–decode–execute.* Esse ciclo contínuo permite que as instruções contidas na ROM sejam lidas, interpretadas e executadas de forma sequencial, assegurando que o sistema opere corretamente.

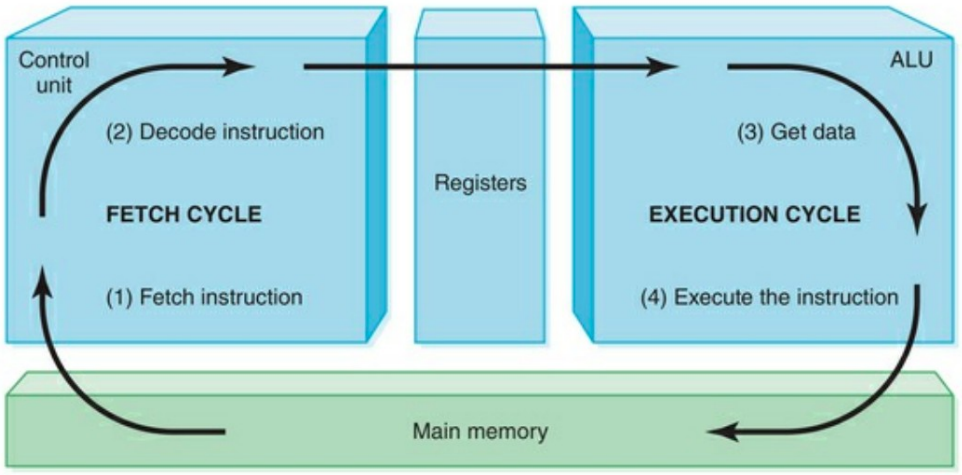
2.2 O CICLO FETCH–DECODE–EXECUTE

O ciclo *fetch–decode–execute* constitui o mecanismo fundamental da arquitetura de von Neumann. Conforme detalhado por (TANENBAUM; AUSTIN, 2013), o processo inicia-se na etapa de *fetch* (busca), onde a CPU recupera a instrução da memória no endereço apontado pelo *Program Counter* (PC) e a carrega para o *Instruction Register* (IR). O PC é então incrementado para apontar para a próxima instrução.

Na etapa subsequente, o *decode* (decodificação), a instrução contida no IR é analisada para determinar qual operação deve ser realizada e quais operandos são necessários. No contexto de um emulador, isso geralmente envolve a extração de bits específicos do *opcode* (código de operação) para rotear o fluxo para a rotina correta. Este termo consiste no conjunto de bits responsável por identificar o tipo de instrução a ser executada. Por fim, na etapa de *execute*, a operação é efetivada: sinais de controle acionam o caminho de dados, podendo envolver cálculos na Unidade Lógica e Aritmética (ALU), movimentação de dados entre registradores ou operações de E/S.

A Figura 2 apresenta uma representação esquemática do ciclo, mostrando como as três etapas se conectam de forma contínua para assegurar a execução correta do *software* original.

Figura 2 – Representação esquemática do ciclo *Fetch–Decode–Execute*



Fonte: DALE; LEWIS (2018).

Garantir a fidelidade na interpretação e execução das instruções é essencial para que o emulador reproduza o comportamento do *software* como ocorreria no *hardware* original, mantendo a integridade da experiência de uso do sistema *target*.

2.3 A IMPORTÂNCIA DO TIMING NA EMULAÇÃO

Apesar da aparente simplicidade do ciclo fetch–decode–execute, sua execução não pode ocorrer de forma arbitrária. A sincronia precisa é essencial para replicar o comportamento do sistema e garantir que os programas funcionem conforme previsto. Essa precisão é determinada principalmente pelo ciclo de máquina, que atua como o “batimento cardíaco” do sistema, definindo o ritmo de execução das instruções (EMULATION GENERAL WIKI, 2024).

O desafio surge porque videogames e máquinas de arcade foram projetados para operar como “máquinas em tempo real”, em que o desempenho dos jogos e programas depende diretamente da exatidão temporal. Para que a experiência seja fiel ao sistema original, o emulador precisa respeitar rigorosamente a cadência com que cada instrução é processada.

Nos sistemas originais, os programadores exploravam ao máximo o hardware disponível, otimizando cada ciclo de processamento para alcançar o desempenho

ideal. As limitações de memória e poder de processamento levaram a soluções criativas que resultaram em mundos imersivos e jogabilidades inovadoras. Exemplos notáveis incluem a névoa em *Silent Hill* e as caixas em *Crash Bandicoot* (SCHULTZ, 2024). Entretanto, essas técnicas geraram jogos e aplicações fortemente dependentes da sincronização em tempo real de áudio, vídeo e controles.

Por isso, variações na velocidade de execução podem comprometer significativamente a experiência: se o emulador rodar rápido demais, o jogo pode se tornar injogável devido à aceleração dos eventos; se for lento, podem ocorrer atrasos perceptíveis, perda de fluidez e comprometimento da interação. Garantir essa sincronização é, portanto, fundamental para que o sistema emulado funcione corretamente, preservando tanto a jogabilidade quanto a autenticidade da experiência original.

2.4 EMULAÇÃO EM ALTO E BAIXO NÍVEL

A emulação de um sistema tem como um dos seus principais objetivos replicar o funcionamento da CPU do sistema target, uma vez que este componente é responsável por executar as instruções dos programas. Para alcançar esse objetivo, são utilizadas diferentes abordagens, que podem ser classificadas em dois níveis principais: **interpretação**, que se concentra na reprodução direta das instruções, e **tradução binária**, que busca otimizar a execução ao traduzir instruções para o sistema host (SANCHEZ, 2018).

A interpretação, também conhecida como emulação de alto nível (*High-Level Emulation* - HLE), é o método mais simples para emular uma CPU. Esse método de emulação é intuitivo e fácil de implementar, pois simula diretamente o comportamento da CPU, lendo e executando as instruções conforme elas aparecem na memória da máquina emulada. A implementação mais comum para isso é o uso de estruturas como *switch/case*, onde cada caso representa um *opcode*.

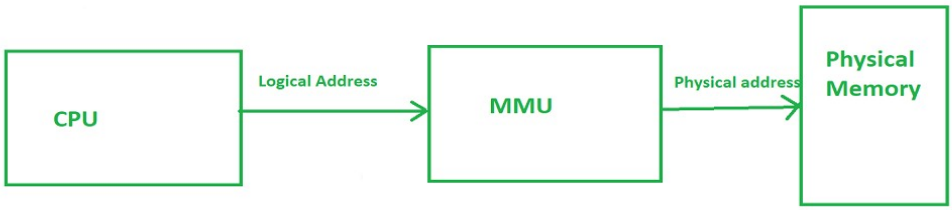
Embora seja uma abordagem direta, a interpretação pode apresentar limitações de desempenho, especialmente em sistemas mais complexos. O processo de decodificação e execução de cada instrução a cada ciclo pode resultar em uma sobrecarga significativa. Esse método não aproveita de forma eficiente as capacidades do sistema *host*, o que pode afetar a performance, principalmente em emulações que necessitam de alta precisão temporal, como na emulação de

sistemas mais recentes.

Por outro lado, a tradução binária, ou emulação de baixo nível (*Low-Level Emulation* - LLE), é uma técnica mais avançada que envolve converter as instruções da CPU emulada para instruções nativas do sistema *host*. Essa abordagem permite que as instruções sejam executadas de forma mais eficiente. Além disso, o código traduzido pode ser reutilizado em múltiplas execuções subsequentes, evitando a necessidade de reprocessamento constante.

No entanto, é importante ressaltar que a performance de um emulador não depende exclusivamente do nível de emulação escolhido, mas também da forma como os conceitos da máquina *target* são mapeados. Um exemplo disso é a emulação da Unidade de Gerenciamento de Memória (*Memory Management Unit* - MMU), que traduz endereços virtuais para endereços físicos na memória. A Figura 3 apresenta um diagrama simplificado da MMU e sua interação com a CPU e a memória física.

Figura 3 – Representação da MMU



Fonte: GEEKFORGEEKS (2024).

Quando essa função é mapeada para a tradução binária, surgem custos significativos de desempenho, pois a tradução de endereços ocorre constantemente a cada acesso à memória. Uma solução utlizada para mitigar esse problema é utilizar as capacidades do sistema host diretamente, rodando o *software* em uma máquina virtual. Isso elimina a necessidade da dispendiosa tradução de endereços, proporcionando ganhos substanciais de desempenho (SANCHEZ, 2018).

Dessa forma, muitos emuladores utilizam uma combinação das abordagens de emulação de alto e baixo nível para otimizar o desempenho. Essa combinação

permite equilibrar a precisão e a eficiência, utilizando as vantagens de cada técnica conforme a necessidade de emulação

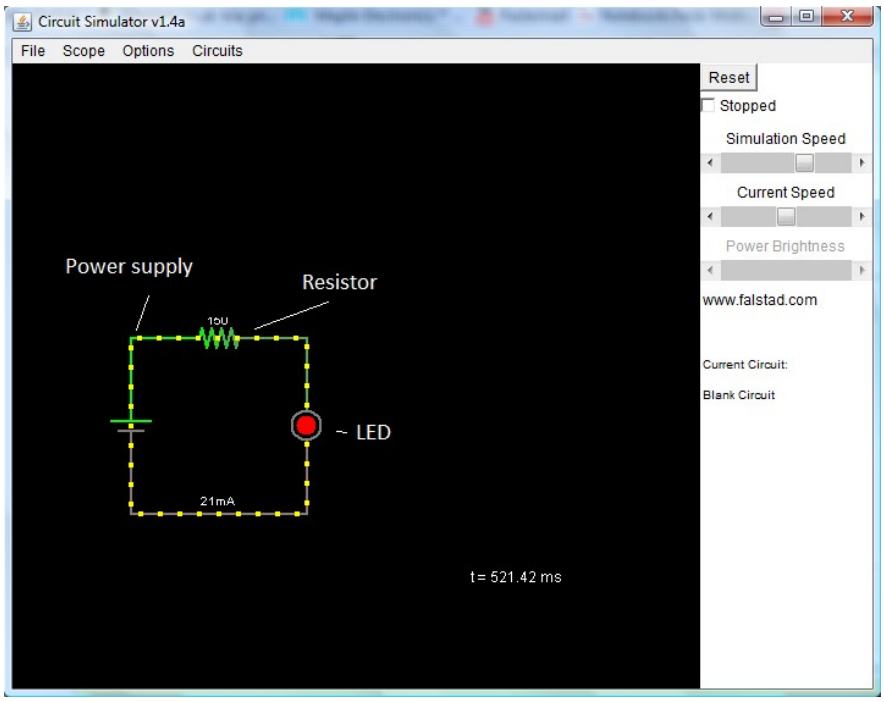
2.5 DIFERENÇAS ENTRE EMULAÇÃO E SIMULAÇÃO

Os termos “emulador” e “simulador” são frequentemente usados de forma intercambiável, o que pode gerar confusão, pois ambos se referem a sistemas que reproduzem o comportamento de outros. No entanto, suas finalidades e níveis de fidelidade diferem significativamente, tornando essencial compreender essas distinções para evitar equívocos (AVENGA, 2022).

A principal diferença reside no grau de precisão com que cada sistema reproduz o original e no objetivo que busca atingir. Enquanto os emuladores têm como foco replicar de forma exata um sistema completo — incluindo hardware e *software* — os simuladores se concentram em reproduzir aspectos ou comportamentos específicos, priorizando flexibilidade em detrimento da fidelidade absoluta.

Simuladores permitem recriar partes selecionadas de um sistema com o objetivo de treinar, testar ou estudar processos e funcionalidades.Por exemplo, em um simulador de circuitos eletrônicos como o Falstad, é possível montar, testar e modificar circuitos em condições ideais, sem o risco de danificar fisicamente os componentes. Essa flexibilidade possibilita a experimentação de cenários hipotéticos, preparando o usuário para diversas situações sem os riscos associados ao sistema original. A Figura 4 ilustra o uso do Falstad, destacando a interface de simulação de circuitos eletrônicos.

Figura 4 – Representação de circuito no Falstad



Fonte: INSTRUCTABLES (2009).

Apesar do Falstad oferecer uma representação detalhada do circuito, é crucial reconhecer que ele não substitui a experiência de um circuito físico real. Afinal, a simulação, por mais precisa que seja, opera em um ambiente incompleto, sem todas as nuances do mundo real. Em contrapartida, os emuladores se esforçam para oferecer uma experiência fiel o suficiente que permita até mesmo a substituição pelo real. Assim, embora compartilhem a proposta de reproduzir sistemas, emuladores e simuladores se distinguem por suas características e finalidades.

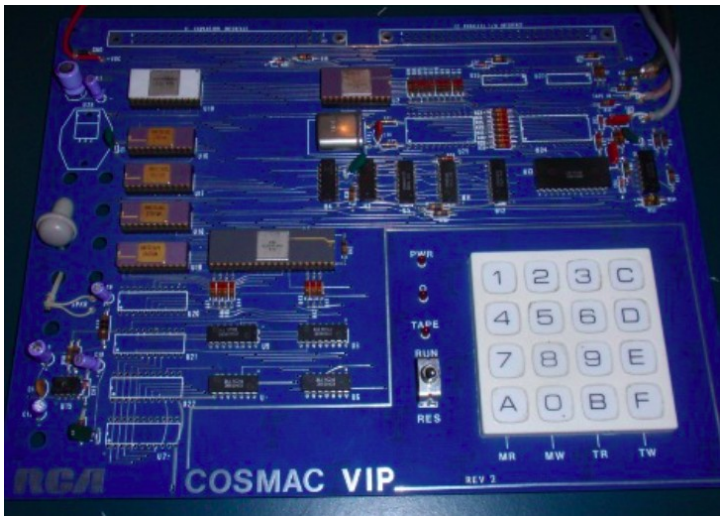
Além dessas duas abordagens, destaca-se ainda o conceito de port, que corresponde a uma forma distinta de adaptação de *software* para novas plataformas. Essa técnica envolve a reescrita ou modificação do código-fonte, permitindo que o programa seja executado nativamente no novo ambiente (GLEN, 2020). Essa abordagem é amplamente utilizada no relançamento de jogos clássicos em plataformas modernas, proporcionando maior eficiência e compatibilidade, sem comprometer a essência do produto original.

2.6 PANORAMA HISTÓRICO DO CHIP-8

A década de 1970 marcou o início da era dos microcomputadores pessoais, um período em que a tecnologia começou a se tornar mais acessível, mas ainda estava longe de ser simples de usar. Os computadores eram limitados em memória, capacidade de processamento e recursos gráficos, o que tornava a criação de jogos e programas um verdadeiro desafio. Desenvolver qualquer aplicação exigia dos programadores grande atenção aos detalhes e profundo conhecimento do hardware, sendo necessário otimizar cada instrução para que os programas funcionassem corretamente. Foi nesse contexto de limitações técnicas e crescente interesse por entretenimento digital que Joseph Weisbecker desenvolveu o CHIP-8, apresentado na revista BYTE em dezembro de 1978 (WEISBECKER, 1978b). Seu nome é um acrônimo para *Comprehensive Hexadecimal Interpretive Programming 8-Bit*, refletindo sua natureza como uma linguagem interpretada voltada para programação simplificada em sistemas de 8 bits. Ele foi criado com o objetivo de facilitar o desenvolvimento de jogos e aplicativos, sendo originalmente projetado para rodar em microcomputadores baseados no processador RCA COSMAC CDP1802. Um dado interessante sobre esse processador é que, devido à sua alta resistência à radiação e baixo consumo de energia, ele foi amplamente utilizado em missões espaciais da NASA e também em diversas aplicações militares, incluindo a utilização no Telescópio Espacial Hubble (XAPSOS et al., 2004).

O CHIP-8 revolucionou o cenário de desenvolvimento de jogos ao introduzir uma linguagem simplificada, que era interpretada pelo sistema e traduzida em tempo real para a execução de jogos e programas. A Figura 5 apresenta o COSMAC VIP, o primeiro computador onde o CHIP-8 foi implementado. Este equipamento possuía 2 KB de RAM, um sistema operacional de 512 bytes e 20 jogos pré-carregados, todos eles programados em CHIP-8 (OLDCOMPUTERS.NET, 2004).

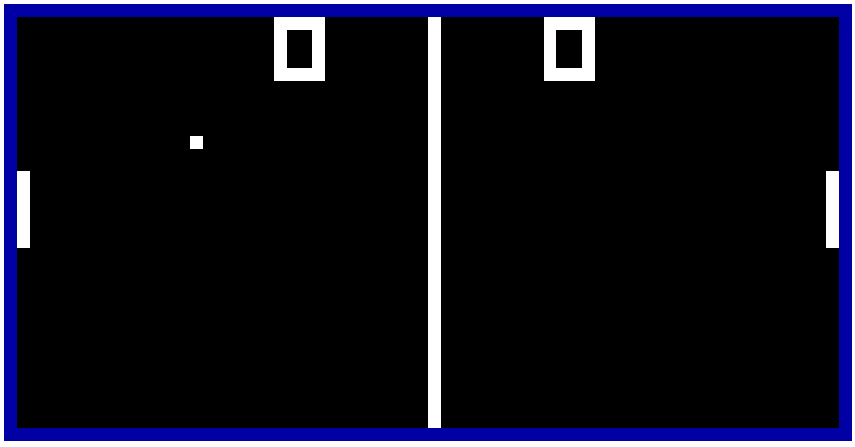
Figura 5 – COSMAC VIP



Fonte: OLDCOMPUTERS.NET (2004).

A simplicidade proporcionada pelo CHIP-8 permitia recriar versões simplificadas de jogos populares da época, como *Pong*, *Tetris* e *Space Invaders*, tornando o aprendizado e a experimentação mais acessíveis para desenvolvedores iniciantes. A Figura 6 apresenta o jogo *Pong*, um dos títulos mais conhecidos implementados em CHIP-8.

Figura 6 – Jogo *Pong*, um dos títulos clássicos recriados no CHIP-8



Fonte: WINTER (1999).

Apesar de sua inovação, o CHIP-8 não alcançou grande popularidade no mercado, pois os computadores COSMAC VIP enfrentavam forte concorrência de microcomputadores mais poderosos, como *Apple II, Atari 400/800, Commodore PET* e *Tandy TRS-80*. Esses sistemas ofereciam maior capacidade de memória, processamento e recursos gráficos, reduzindo o interesse pelo CHIP-8 e limitando sua difusão inicial (ATARIWIKI, 2024). Ainda assim, a plataforma se manteve relevante para experimentação e aprendizado, servindo de base para o desenvolvimento de emuladores modernos e recriações em diferentes sistemas.

Ao longo dos anos, ele deu origem a várias versões modificadas, cada uma contribuindo para a evolução da plataforma e ampliando suas capacidades (MIKOLAY, 2017). Entre as mais relevantes, destacam-se:

- *CHIP-48:* Criado em 1990 por Andreas Gustafsson para a linha de calculadoras *HP-48,* o CHIP-48 representou uma adaptação do CHIP-8 para o hardware de calculadoras, que na época possuíam recursos mais limitados do que os computadores pessoais.
- *SUPER-CHIP (SCHIP):* Desenvolvido em 1991 por Erik Bryntse, o S*UPER-CHIP* aprimorou o CHIP-48 ao adicionar um modo gráfico de alta resolução e instruções para rolagem de tela. Sua popularidade foi notável, rapidamente substituindo o CHIP-48 como a variante mais utilizada.
- *XO-CHIP:* Criado em 2014 por John Earnest, o *XO-CHIP* é uma variante moderna que surgiu junto com o emulador Octo. Ele incorpora diversas funções avançadas, como sistema de áudio mais complexo, suporte a 64 KB de memória e múltiplos planos gráficos, expandindo consideravelmente as capacidades do CHIP-8 original.

Após conhecer as principais variantes, é importante esclarecer um ponto sobre a forma como esses sistemas funcionam. Embora sejam comumente chamados de “emuladores”, os programas CHIP-8 não foram desenvolvidos para rodar diretamente em hardware físico específico, mas sim sobre uma máquina virtual — uma camada de abstração que simula o ambiente de execução (ROLINDAW, 2024). Na documentação original, Joseph Weisbecker emprega tanto os termos “registrador” quanto “variável” para se referir a componentes internos, reforçando que o CHIP-8 opera em um nível acima do hardware real (WEISBECKER, 1978b).

O uso do termo “emulador” se consolidou principalmente pela função prática desses sistemas: recriar a experiência de execução de programas CHIP-8, originalmente criados para o COSMAC VIP. Assim, o que é efetivamente reproduzido é a lógica e o comportamento da arquitetura virtual, permitindo que os programas se comportem da mesma forma que no ambiente original.

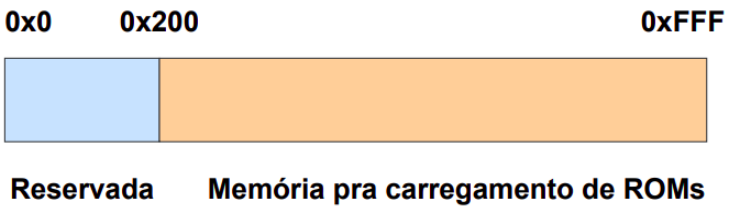
Embora essa distinção técnica seja importante para uma compreensão mais precisa, ela não diminui a relevância do CHIP-8 em contextos educacionais e de estudo de emulação. As abstrações utilizadas seguem princípios semelhantes aos de arquiteturas físicas, como a interpretação de instruções, a atualização de registradores e o gerenciamento de temporizadores, oferecendo um ambiente de aprendizado que reproduz desafios típicos de processadores reais. Dessa maneira, o CHIP-8 funciona como uma introdução acessível ao estudo de emulação e ao entendimento do funcionamento interno de sistemas computacionais.

O interesse pelo sistema permanece ativo até hoje, com entusiastas desenvolvendo projetos que recriam sua arquitetura original, inclusive em protótipos físicos (CHIP-8.COM, 2018). Esse contínuo envolvimento evidencia o impacto duradouro do CHIP-8 na história da computação e seu valor como ferramenta pedagógica.

2.7 ARQUITETURA DO CHIP-8

A arquitetura do CHIP-8 é baseada em um design minimalista, porém funcional, incorporando componentes essenciais para o seu funcionamento. A memória do sistema consiste em 4 KB (4096 bytes) de RAM, organizada de forma estruturada para atender diferentes necessidades. Os primeiros 512 bytes, correspondentes aos endereços de 0x000 a 0x1FF (0 a 511), são reservados para o interpretador, abrigando rotinas essenciais do sistema e a representação das fontes hexadecimais utilizadas na exibição de caracteres na tela. Embora os dados das fontes possam ser armazenados em qualquer posição dentro desse intervalo, é comum seguir a convenção de alocá-los entre os endereços 0x050 (80) e 0x09F (159) (LANGHOFF, 2020). A partir do endereço 0x200 (512), inicia-se o carregamento dos programas na memória, garantindo que o espaço destinado ao interpretador permaneça protegido durante a execução dos programas. A Figura 7 ilustra a organização da memória do CHIP-8.

Figura 7 – Separação da Memória



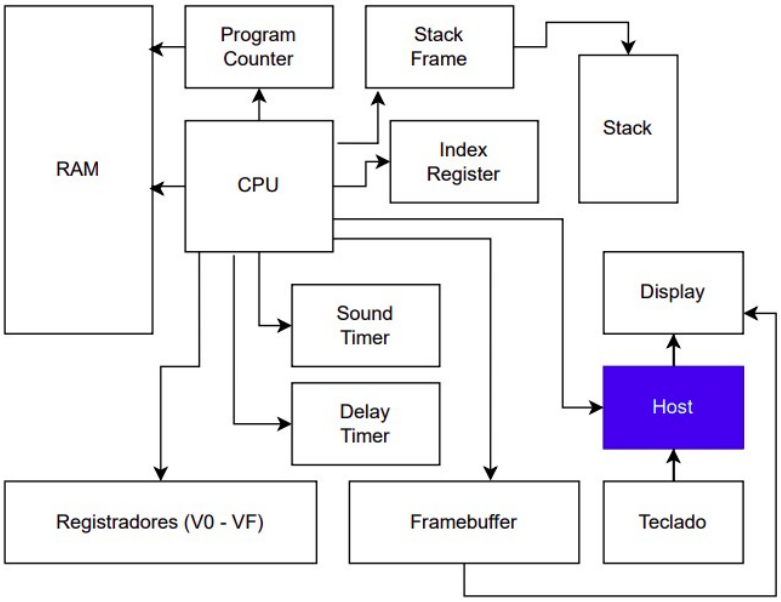
Fonte: Elaborado pelo Autor

Em relação aos registradores, o CHIP-8 dispõe de 16 registradores de uso geral de 8 bits, identificados de V0 a VF. Esses registradores armazenam dados temporários durante a execução das instruções, facilitando o processamento e a manipulação de informações pelo sistema. O registrador VF possui uma função especial, atuando como uma flag de status em determinadas operações, sendo frequentemente utilizado para indicar eventos como colisões gráficas na tela durante a renderização.

O CHIP-8 também conta com um registrador de índice (I) de 16 bits, utilizado para armazenar endereços de memória, e um contador de programa (PC), também de 16 bits, responsável por apontar para a próxima instrução a ser executada, garantindo o fluxo sequencial do programa. Ambos possuem 16 bits para permitir o acesso ao endereço máximo da memória (0xFFF). Para controle de fluxo em sub-rotinas, o sistema utiliza uma pilha com capacidade para 16 níveis, capaz de armazenar 16 valores de 16 bits, acompanhada de um ponteiro de pilha (*Stack Pointer* - SP) de 8 bits.

A Figura 8 apresenta um diagrama de blocos do CHIP-8, ilustrando a interação entre os diversos componentes do sistema.

Figura 8 – Diagrama de blocos do sistema

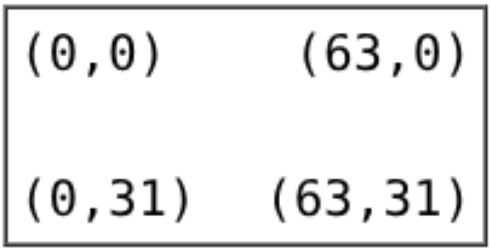


Fonte: Elaborado pelo Autor

2.7.1 Processo de Renderização Gráfica

A interface gráfica do CHIP-8 é baseada em um display monocromático com resolução de 64x32 pixels e é realizada por meio de um sistema de desenho baseado em *sprites*. *Este termo* pode ser entendido como um conjunto de pixels que representa um objeto gráfico na tela, como um personagem, caractere ou elemento visual do jogo. No contexto do CHIP-8, esses *sprites* são definidos como matrizes binárias, em que cada bit corresponde ao estado de um pixel (ativado ou desativado).

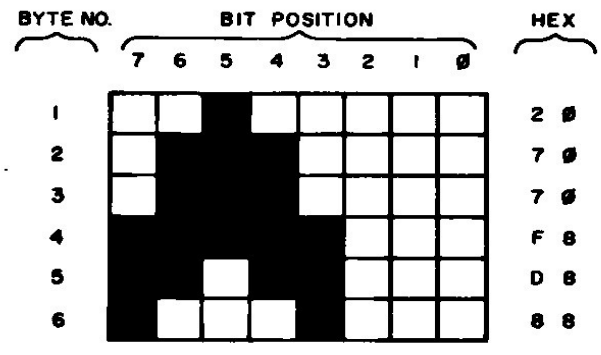
Figura 9 – Tela do CHIP-8



Fonte: GREENE (1997)

A instrução DXYN é responsável pela renderização de sprites na tela. Nessa instrução, X e Y referem-se aos registradores Vx e Vy, que armazenam, respectivamente, as coordenadas x e y da posição onde o sprite será desenhado na tela. O parâmetro N define a altura do *sprite* em bytes, sendo que cada byte representa uma linha de 8 pixels de largura. Por exemplo, se N = 5, o sistema interpreta o sprite como um dígito hexadecimal (4x5 pixels). Já se N = 8, o sprite é renderizado como um padrão genérico de 8x8 pixels. Assim, a combinação dos valores em Vx, Vy e N determina a posição e o tamanho do sprite na tela (SCOTFORD, 2020a). O registrador I é responsável por sinalizar o endereço na memória onde o padrão do sprite está armazenado. Esse padrão é basicamente uma sequência de bytes que formam a imagem. A Figura 10 ilustra um exemplo de um sprite genérico de um foguete.

Figura 10 – Representação de um *sprite* de foguete



Fonte: WEISBECKER (1978c)

O desenho dos *sprites* no display do CHIP-8 é feito utilizando uma operação lógica chamada XOR, ou "ou exclusivo". Na prática, isso significa que cada pixel muda de estado: se estava apagado (0), passa a ser ativado (1); se estava ativo, é apagado. Essa técnica permite sobrepor e remover elementos gráficos sem precisar limpar toda a tela, tornando o processo mais rápido e eficiente. Por exemplo, ao mover um objeto, basta reaplicar a operação XOR nas mesmas coordenadas para apagá-lo antes de desenhá-lo em outro lugar. Além disso, ela facilita a detecção de colisões: se um pixel é apagado por sobreposição, o registrador de flag VF é definido como 1, indicando que houve colisão.

Os pixels são normalmente armazenados em uma área de memória chamada ***framebuffer***, que funciona como um mapa de bits representando o estado atual de cada ponto da tela (PRADA, 2009). Ao desenhar um sprite, todas as alterações são feitas primeiro no *framebuffer*, que depois é usado para atualizar a tela. Esse método garante que a renderização seja suave e contínua, evitando piscadas ou atrasos na exibição dos gráficos.

2.7.2 Subsistema de Áudio e Timers

O CHIP-8 possui dois timers independentes que decrementam a uma taxa fixa de 60 Hz. O primeiro, denominado *Delay Timer* (DT), é utilizado para controlar a duração de determinados eventos temporais. O segundo, chamado *Sound Timer* (ST), emite um sinal sonoro enquanto seu valor for diferente de zero, funcionando como um alerta para o usuário.

O controle dos timers é realizado por meio da instrução FX18, em que o valor do registrador VX define a duração do som emitido. O *Sound Timer* decrementa seu valor a uma taxa fixa de 60Hz, ou seja, reduz uma unidade a cada 1/60 de segundo (aproximadamente 16,7ms). Dessa forma, a duração total do som depende diretamente do valor inicial armazenado em VX. Por exemplo, se VX for definido como FF (255 em decimal), o timer precisará contar de 255 até 0, levando cerca de 4,25 segundos (255 ciclos × 16,7ms). Valores menores de VX resultam em tempos de som proporcionalmente reduzidos, sendo o intervalo mínimo prático equivalente a 2 (SCOTFORD, 2020b). Essa abordagem permite que o CHIP-8 produza sons de durações variadas com apenas um registrador, demonstrando como a simplicidade da arquitetura ainda possibilita controle preciso de eventos temporais.

2.7.3 Mapeamento do Teclado

Em relação à entrada de dados, o CHIP-8 utiliza um teclado hexadecimal composto por 16 teclas, numeradas de 0 a F. Cada tecla representa um valor hexadecimal de 4 bits, e o sistema monitora continuamente as entradas do usuário para executar comandos conforme as instruções do programa em execução. Embora simples, esse teclado é essencial para a interação do usuário com os jogos e programas desenvolvidos para o CHIP-8.

Como os teclados modernos não possuem um layout hexadecimal específico, é comum adaptar esse esquema para algo mais familiar, como o teclado *QWERTY*. A Figura 11 apresenta um dos mapeamentos mais utilizados para essa adaptação.

Figura 11 – Mapeamento das teclas do CHIP-8 para um teclado *QWERTY*

Teclado do CHIP-8

1	2	3	C
4	5	6	D
7	8	9	E
A	0	B	F

Teclado do Emulador

1	2	3	4
Q	W	E	R
A	S	D	F
Z	X	C	V

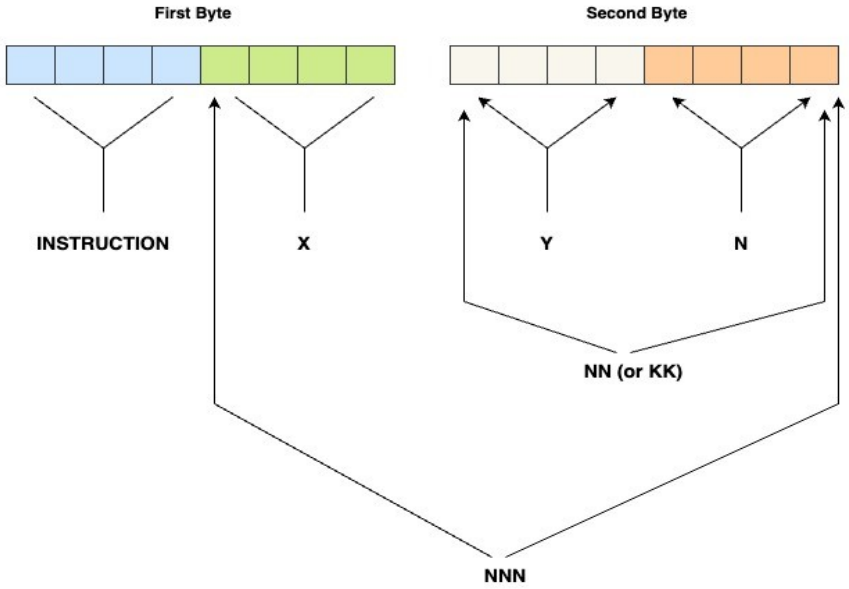
Fonte: Elaborado pelo Autor.

2.8 ESTRUTURA DAS INSTRUÇÕES

Para compreender o funcionamento do sistema, é fundamental analisar a estrutura de suas instruções. Cada instrução é composta por dois bytes (16 bits) e organiza as informações em duas partes principais: o *opcode* e os operandos. O *opcode*, representado pelos primeiros 4 bits (um *nibble*), determina qual operação deve ser executada, enquanto os 12 bits restantes são reservados aos operandos,

fornecendo os dados ou endereços necessários para a operação correspondente. A forma como os operandos são interpretados varia conforme o *opcode*, podendo indicar registradores, valores imediatos ou endereços de memória. A Figura 12 apresenta um esquema visual que ilustra a distribuição desses bits dentro de uma instrução típica.

Figura 12 – Estrutura de uma instrução



Fonte: VALADARES (2024).

Considerando a instrução 0x6402, é possível decompor seus componentes da seguinte forma: os primeiros 4 bits (ou o primeiro *nibble*), 0x6, representam o *opcode*. Nesse contexto, o *opcode* 0x6 indica a operação de carregar um valor imediato em um registrador (GREENE, 1997). O próximo *nibble*, 0x4, identifica o registrador alvo, neste caso, o registrador V[4]. Por fim, os 8 bits restantes, 0x02, correspondem ao valor imediato que será carregado em V[4]. Assim, o valor final carregado é 0x02 (ou 2 em decimal).

Portanto, a instrução 0x6402 realiza a operação V[4] = 0x02, ou seja, carrega

o valor 0x02 no registrador V[4]. Esse exemplo evidencia como a instrução é estruturada: o *opcode* determina a operação, enquanto os operandos especificam os detalhes da execução.

O conjunto de instruções original possuía 31 operações distintas, cada uma com lógica específica para a interpretação dos bits. No entanto, o número total de instruções é, na prática, 36, pois cinco *opcodes* estavam localizados na ALU do microprocessador *CDP-1802,* configurando funcionalidades que possivelmente não eram intencionais (WEISBECKER, 1978a).

Em comparação com arquiteturas modernas, como x86 e ARM, que empregam modos de endereçamento complexos, a arquitetura minimalista do CHIP-8 se destaca. Essa simplicidade facilita a compreensão de conceitos fundamentais da computação, como o ciclo *fetch-decode-execute,* a manipulação de registradores e o desenvolvimento de emuladores. Ao reduzir a complexidade desnecessária, o sistema permite que estudantes e iniciantes foquem nos princípios básicos, criando uma base sólida para o estudo de arquiteturas mais avançadas.

2.8.1 Imprecisões da Documentação Tradicional

A documentação mais conhecida sobre o CHIP-8, elaborada por Cowgod, serviu por muitos anos como referência principal para a implementação de emuladores. No entanto, análises modernas identificaram diversas imprecisões que merecem destaque.

Entre os erros mais relevantes estão: a ordem incorreta de aplicação dos flags aritméticos nas instruções da ALU; interpretações equivocadas sobre a instrução gráfica Dxyn e o tratamento de colisões de pixels; e a afirmação incorreta de que os sprites que ultrapassam os limites da tela deveriam se repetir no lado oposto. Na prática, o comportamento correto consiste no recorte da área que excede os limites da tela.

Outra instrução cuja interpretação foi corrigida é a Fx0A, que, ao contrário do indicado por Cowgod, aguarda a liberação de uma tecla, e não apenas a sua pressão inicial. Essas correções resultam de investigações e testes empíricos conduzidos pela comunidade de emulação, evidenciando o esforço coletivo em refinar e preservar a precisão técnica do CHIP-8 (8924TH, 2023).

3 TRABALHOS RELACIONADOS

Com o avanço das tecnologias digitais, o CHIP-8 deu origem a uma ampla variedade de emuladores em diferentes plataformas, permitindo que entusiastas revivam jogos clássicos de maneira acessível. A simplicidade do sistema e a facilidade de implementação tornaram-no uma escolha popular entre desenvolvedores amadores e aqueles interessados em explorar a programação de emuladores.

Apesar disso, muitos desses emuladores apresentam uma experiência limitada em termos de funcionalidades. Embora a fidelidade ao sistema original seja essencial para preservar a essência do CHIP-8 e dos jogos criados para ele, poucos aproveitam os avanços tecnológicos disponíveis, que poderiam tornar a experiência mais imersiva e interativa para jogadores modernos.

Recursos como melhorias na interface, integrações com tecnologias atuais e adição de funções modernas poderiam enriquecer significativamente a experiência de emulação. No entanto, essas possibilidades frequentemente não são exploradas, restringindo a evolução do uso e das experiências proporcionadas pelos emuladores.

Por outro lado, ao replicarem fielmente o funcionamento do sistema original, esses emuladores cumprem um papel fundamental na preservação da memória histórica dos primeiros jogos desenvolvidos para o CHIP-8, garantindo que a experiência nostálgica continue acessível às novas gerações de usuários.

3.1 OCTO

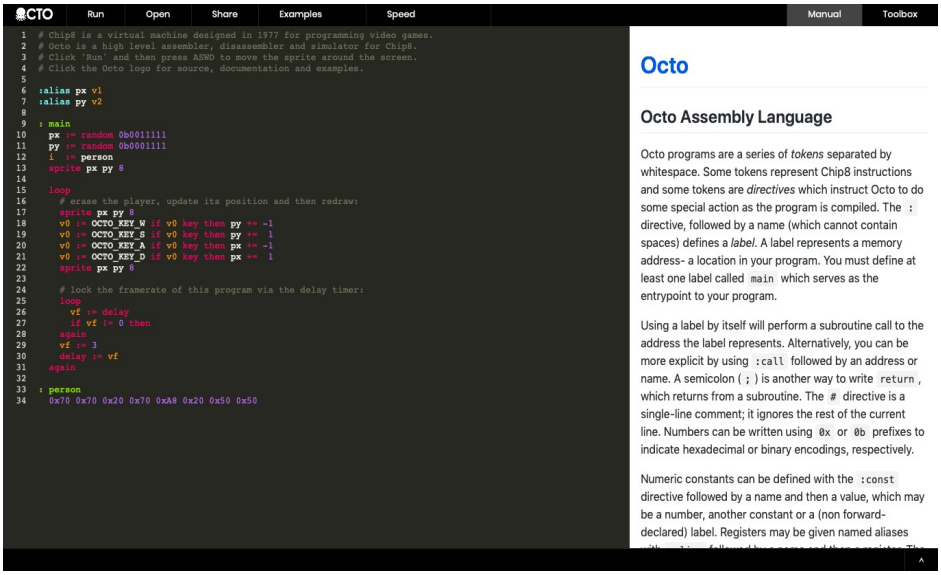
O *Octo* é um dos emuladores mais conhecidos do CHIP-8, desenvolvido por John Earnest e projetado para funcionar diretamente no navegador. Apesar de oferecer um ambiente de emulação, seu foco principal está no desenvolvimento de novos programas para essa plataforma (EARNEST, 2013).

Ele disponibiliza um ambiente integrado de desenvolvimento (IDE), permitindo que desenvolvedores criem seus próprios jogos e aplicativos para CHIP-8 em um único local, utilizando uma linguagem assembly simplificada e acessível. Além disso, existe um site dedicado a jogos sem direitos autorais desenvolvidos na IDE, chamado *CHIP-8 Archive* (EARNEST, 2023). Esse acervo reúne diversos projetos

criados na plataforma, possibilitando que usuários explorem e experimentem jogos feitos para o interpretador CHIP-8, promovendo o compartilhamento de conhecimento e preservando a comunidade em torno dessa plataforma histórica.

A Figura 13 apresenta a tela inicial do *Octo*, mostrando como os usuários podem acessar facilmente suas funcionalidades de emulação e desenvolvimento.

Figura 13 – Tela inicial do *Octo*



Fonte: EARNEST (2013).

3.2 CHIP8.JS

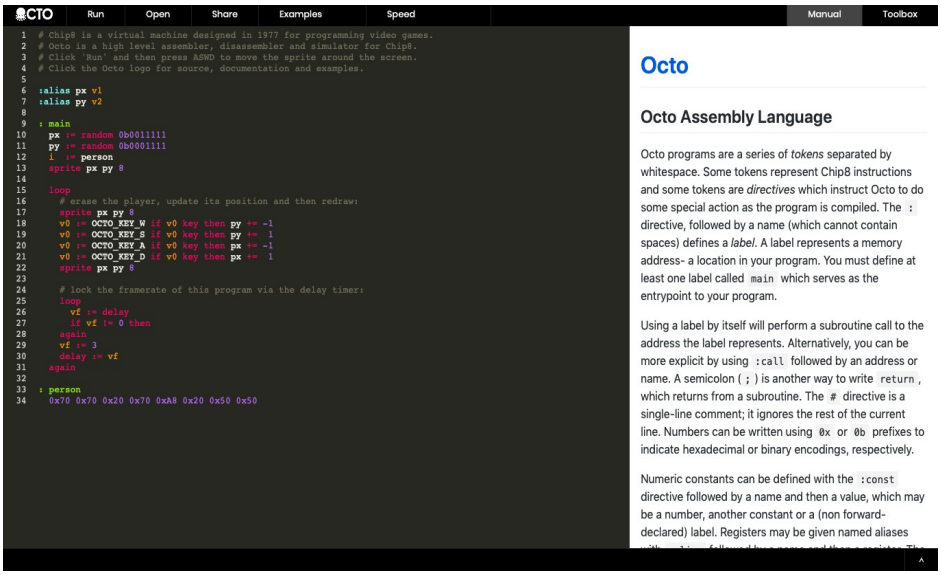
O Chip8.js é um emulador de CHIP-8 desenvolvido em JavaScript, projetado para ser executado diretamente em navegadores web. Inspirado na estética retrô, ele recria a experiência nostálgica dos jogos clássicos de 8 bits, utilizando cores pixelizadas e fontes características da época. O objetivo principal é proporcionar uma experiência acessível e intuitiva, sem a necessidade de instalações ou configurações complexas.

O emulador inclui uma lista pré-definida de jogos, permitindo a execução imediata de programas CHIP-8 sem dependências externas. Essa abordagem facilita a exploração do sistema para iniciantes que desejam experimentar os jogos clássicos de forma rápida e prática. No entanto, essa simplicidade traz algumas limitações. O Chip8.js não permite carregar ROMs personalizadas dinamicamente,

restringindo-se aos jogos disponíveis na lista pré-definida. Dessa forma, usuários que desejam criar ou testar seus próprios programas podem encontrar limitações na personalização do ambiente.

Comparado a outros emuladores mais flexíveis, que oferecem recursos avançados e suporte para ROMs externas, o Chip8.js não é a melhor opção para usuários que buscam exploração mais profunda. Ainda assim, para quem procura uma experiência rápida, direta e sem complicações técnicas, ele se apresenta como uma alternativa eficiente. A Figura 14 ilustra a tela inicial do Chip8.js, destacando sua interface minimalista e fiel à estética retrô.

Figura 14 – Tela inicial do Chip8.js



Fonte: RASCIA (2021).

3.3 OUTRAS IMPLEMENTAÇÕES RELEVANTES

Embora existam diversos projetos que emulam o CHIP-8, a maior parte deles apresenta funcionalidades bastante limitadas (EMULATION GENERAL WIKI, 2025). Em geral, esses emuladores se restringem à reprodução do comportamento básico do sistema, sem fornecer informações ou recursos que permitam ao usuário compreender detalhadamente seu funcionamento.

Dessa forma, embora atendam ao propósito de executar programas e jogos clássicos, tais implementações carecem de mecanismos didáticos ou explicativos

que facilitem a aprendizagem sobre a arquitetura e operação do CHIP-8. Considerando essas limitações, o presente trabalho propõe ampliar as possibilidades oferecidas pelo CHIP-8, mantendo sua essência original, mas integrando recursos que favoreçam a compreensão do sistema e a interação do usuário de forma mais completa.

4 METODOLOGIA

Para a realização deste trabalho foi seguida uma sequência de etapas voltadas à pesquisa, análise e implementação do sistema emulador CHIP-8. Durante o desenvolvimento, foram realizadas consultas a materiais técnicos e projetos similares, de forma a orientar a concepção da arquitetura e a fidelidade da emulação.

4.1 PESQUISA BIBLIOGRÁFICA

Foi realizada uma pesquisa bibliográfica com o objetivo de compreender a estrutura e o funcionamento do CHIP-8, incluindo o formato de suas instruções, o papel dos registradores, o uso da memória e o mecanismo de temporização. Além disso, foram analisados projetos open-source de emuladores existentes, a fim de identificar boas práticas de implementação e estratégias de interface que pudessem ser adaptadas ao contexto educacional proposto. Os resultados dessa pesquisa serviram de base para a definição da arquitetura e das funcionalidades do emulador descritas nos capítulos seguintes.

Dentre as diversas variantes do CHIP-8 existentes, optou-se por focar na versão clássica (ou tradicional), conforme documentada nas especificações originais de Joseph Weisbecker (GREENE, 1997). Essa escolha foi motivada pela maior fidelidade histórica, simplicidade de implementação e ampla disponibilidade de programas e documentação compatíveis com o interpretador original do COSMAC VIP. Ao evitar extensões que alteram o comportamento de *opcodes* ou adicionam funcionalidades não presentes no sistema primitivo, garante-se uma emulação mais pura, didática e alinhada aos objetivos pedagógicos do trabalho. Explicar a escolha da versão clássica é fundamental, pois cada variante do CHIP-8 introduz instruções novas ou diferentes, podendo alterar o comportamento de programas e complicar a compreensão do funcionamento original do sistema (GULRAK, 2023).

4.2 DESENVOLVIMENTO E EXECUÇÃO DO EMULADOR

A arquitetura do emulador foi concebida para reproduzir com alta fidelidade as funções internas do CHIP-8, adotando uma abordagem modular que separa de forma clara as responsabilidades de cada componente do sistema. Essa organização segue princípios consolidados de engenharia de *software*, proporcionando maior facilidade de manutenção, depuração e valor pedagógico do código.

A modularidade se reflete na implementação independente de cada subsistema central — como CPU, memória, display e temporizadores —, que se comunicam por meio de interfaces bem definidas. Essa estrutura não apenas facilita a realização de expansões ou modificações futuras sem comprometer o projeto original, como também espelha a lógica funcional de um computador físico, em que cada subsistema desempenha uma função específica e dedicada.

4.2.1 Tecnologias Utilizadas

O projeto foi construído com base em um conjunto de tecnologias modernas, selecionadas pela clareza de sua sintaxe, pela ampla documentação e pela capacidade de expressar conceitos complexos de forma acessível. A escolha dessas ferramentas visou equilibrar simplicidade e precisão técnica, tornando o emulador tanto um recurso didático quanto um exemplo de boas práticas de engenharia de *software*.

4.2.1.1 TypeScript

O TypeScript foi adotado como linguagem principal do projeto. Sua principal vantagem está na tipagem estática e na orientação a objetos, o que garante maior segurança durante o desenvolvimento e reduz a probabilidade de erros em tempo de execução. Essa característica foi especialmente importante para um sistema de emulação, que exige precisão na manipulação de registradores, memória e temporizadores.

Em comparação com linguagens de baixo nível, como C, o TypeScript oferece maior agilidade e uma curva de aprendizado mais suave, sem exigir o gerenciamento manual de memória. Apesar dessa abstração, o projeto manteve o

rigor conceitual característico das implementações tradicionais, respeitando a arquitetura original do CHIP-8 e suas limitações históricas.

4.2.1.2 React

A camada de interface foi implementada utilizando o React, um framework amplamente empregado para construção de interfaces reativas. Sua integração com o TypeScript permitiu a criação de uma estrutura modular, na qual cada componente visual — como tela de exibição, controles e carregamento de ROMs — pôde ser desenvolvido de forma independente e reaproveitável.

A estilização da interface foi realizada com CSS (*Cascading Style Sheets*) modular, garantindo organização e facilidade de manutenção do design. Adicionalmente, foram utilizadas bibliotecas utilitárias que possibilitaram uma aparência limpa, responsiva e compatível com diferentes dispositivos e tamanhos de tela.

O sistema de áudio foi implementado por meio da **Web Audio API,** uma tecnologia nativa dos navegadores modernos que permite a geração de sons em tempo real. Essa ferramenta foi empregada para reproduzir o “beep” característico do CHIP-8 sempre que o temporizador de som é ativado, respeitando as limitações do hardware original e mantendo a fidelidade à experiência do sistema emulado, seguindo as políticas de execução de áudio impostas pelos navegadores.

4.2.2 Arquitetura e Organização do Projeto

Seguindo uma abordagem modular, a implementação foi estruturada em duas partes principais e bem definidas: o Núcleo de Emulação (ou *Core*) e a Interface do Usuário (*User Interface* - UI). Essa separação foi essencial para garantir um desenvolvimento mais organizado, permitindo que cada módulo fosse desenvolvido e testado de forma independente.

4.2.2.1 O Núcleo de Emulação (Core)

O núcleo representa o motor principal do emulador, responsável por executar toda a lógica interna do CHIP-8. Ele foi desenvolvido em TypeScript puro, de forma

independente de qualquer biblioteca gráfica, garantindo sua portabilidade para diferentes ambientes JavaScript. Essa camada concentra os componentes essenciais da emulação e define o comportamento do sistema de maneira totalmente isolada da interface, servindo como a base sobre a qual o restante do emulador opera.

4.2.2.2 A Interface de Usuário (UI – User Interface)

A camada de visualização tem como principal responsabilidade representar, de forma visual e interativa, o estado interno do núcleo de emulação. Ela atua como uma ponte entre a lógica interna e o usuário, exibindo em tempo real as mudanças que ocorrem no sistema — como a atualização do display, a resposta a comandos e o comportamento geral do emulador. Dessa forma, torna-se possível compreender visualmente o funcionamento interno da máquina, reforçando o caráter didático e acessível do projeto.

4.2.3 Arquitetura geral do Sistema

A arquitetura do CHIP-8 pode ser dividida em quatro componentes principais, cada um desempenhando funções específicas essenciais para o funcionamento do sistema:

- **CPU:** encarregada de buscar (*fetch*), decodificar (*decode*) e executar (*execute*) as instruções do programa, garantindo o fluxo correto das operações.
- **Memória:** responsável pelo armazenamento do código do programa e dos dados temporários utilizados durante a execução das instruções, incluindo variáveis, registradores e *sprites*.
- **Display:** sistema de saída gráfica que exibe os *sprites* e informações visuais em uma tela monocromática de 64x32 pixels, permitindo a representação do estado do sistema de forma visual.
- **Teclado:** interface de entrada que possibilita a interação do usuário com o emulador, simulando o teclado hexadecimal original do CHIP-8 e permitindo a execução de comandos definidos pelo programa.

Para facilitar o desenvolvimento, a implementação foi realizada de forma incremental, começando pelos módulos independentes — aqueles que não dependem de outros componentes para funcionar corretamente. Essa abordagem modular permite testar e validar cada parte separadamente antes de integrá-las, garantindo maior controle e clareza no processo de construção.
Por exemplo, a CPU, que atua como o “cérebro” do sistema, depende diretamente da memória para buscar e executar as instruções. Assim, antes de desenvolver a CPU, é necessário que a memória já esteja implementada. Da mesma forma, o display e o teclado, embora se comuniquem com o núcleo, podem ser desenvolvidos isoladamente, pois possuem comportamentos bem definidos e independentes.

4.2.4 Fluxo de Execução do Emulador

O funcionamento do sistema segue um ciclo bem definido, semelhante ao de um processador real. De forma geral, esse fluxo pode ser dividido em três grandes etapas: **inicialização**, **execução** e **exibição**. Cada uma delas desempenha um papel específico no funcionamento do emulador, garantindo que o sistema opere de maneira contínua e sincronizada.

Na fase de inicialização, são criadas as instâncias dos principais componentes do sistema — CPU, memória, display e teclado. Nessa etapa, as fontes de caracteres são carregadas na memória, e o programa (ROM) selecionado pelo usuário é transferido para o endereço 0x200, conforme estabelecido pela especificação original do CHIP-8.

Após essa fase, o sistema entra no ciclo principal de execução, baseado no modelo clássico *fetch-decode-execute*, que define a sequência lógica de execução das instruções.

4.2.4.1 Busca (*Fetch*)

Nesta etapa, a CPU obtém o próximo *opcode* diretamente da memória. O contador de programa (PC) indica o endereço atual da instrução a ser lida. Como cada instrução do CHIP-8 possui 2 bytes, a CPU lê dois endereços consecutivos e, em seguida, incrementa o PC para apontar para a próxima instrução.

4.2.4.2 Decodificação (*Decode*)

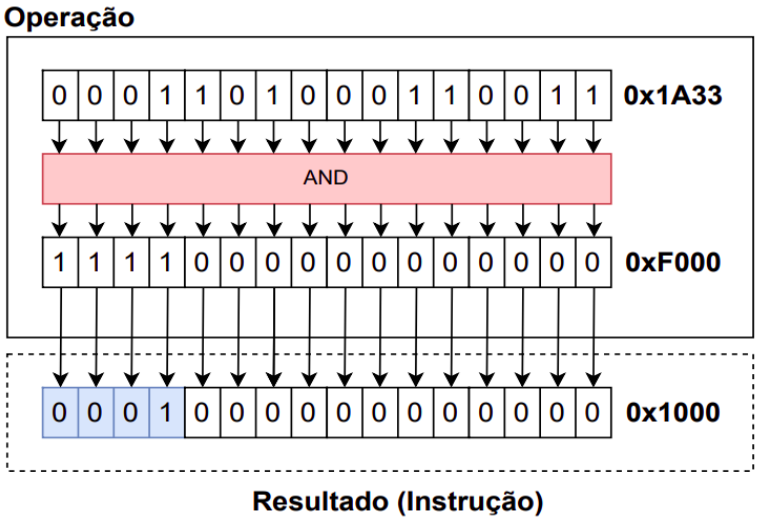
Após a leitura, o *opcode* é decodificado. Essa etapa consiste em identificar o tipo de instrução e os registradores ou constantes envolvidos. O decodificador interpreta o padrão binário do *opcode* e o traduz em uma ação específica, como movimentar dados, realizar operações lógicas ou manipular o display.

Na prática, a decodificação é realizada por meio de operações *bitwise*, ou seja, manipulações em nível de bit. Cada instrução do CHIP-8 é composta por 16 bits e, como mencionado anteriormente, determinadas seções do *opcode* representam o tipo de operação ou operandos.

Na Figura 15, é apresentado um exemplo de decodificação em que o valor 0x1A33 é submetido a uma operação lógica AND com a máscara 0xF000, isolando os 4 bits mais significativos. O resultado dessa operação, 0x1000, corresponde à identificação do tipo de instrução — neste caso, a instrução de salto (JP addr).

Esse mesmo princípio é aplicado às demais partes da instrução, utilizando diferentes máscaras para isolar os campos correspondentes aos registradores e valores imediatos, permitindo que o emulador interprete corretamente cada componente da instrução.

Figura 15 – Exemplo de decodificação de *opcode* por meio de operação lógica AND



Fonte: Elaborado pelo Autor.

4.2.4.3 Execução (*Execute*)

Na fase de execução, a CPU realiza a operação correspondente ao *opcode* decodificado. Isso pode resultar em modificações no estado interno da CPU (como valores em registradores), na memória ou em alterações gráficas no display.

Como exemplo, pode-se citar novamente a instrução de salto (JP addr), representada pelo formato 1NNN, cuja execução faz com que o valor NNN — formado pela combinação dos três últimos *nibbles* — seja carregado no contador de programa (PC). Dessa forma, a execução passa a continuar a partir desse novo endereço de memória, redirecionando o fluxo normal do programa.

4.2.4.4 Atualização (*Update*)

Após a execução da instrução, são atualizados os componentes de tempo do sistema, como os temporizadores de atraso e de som. Caso ocorram alterações visuais, o display é redesenhado. A atualização do display ocorre a partir da instrução DXYN, que transfere os dados de sprite da memória para o *framebuffer*. Esse framebuffer funciona como uma matriz bidimensional de pixels, onde cada posição representa o estado de um ponto da tela (aceso ou apagado).

Assim, sempre que o programa executa uma instrução de desenho, o emulador modifica essa matriz e, em seguida, redesenha o conteúdo no display, refletindo as alterações visuais de forma sincronizada. Assim, embora o CHIP-8 original seja simples e não possua um sistema complexo de interrupções, ele precisa lidar com eventos externos. Entre eles estão a captura das entradas do teclado, a atualização do estado do display e a emissão de sons quando o temporizador de áudio atinge um valor diferente de zero.

Esses eventos são verificados continuamente a cada ciclo de execução, permitindo que o programa responda dinamicamente às ações do usuário sem interromper o fluxo principal.

4.2.5 Validação e Testes

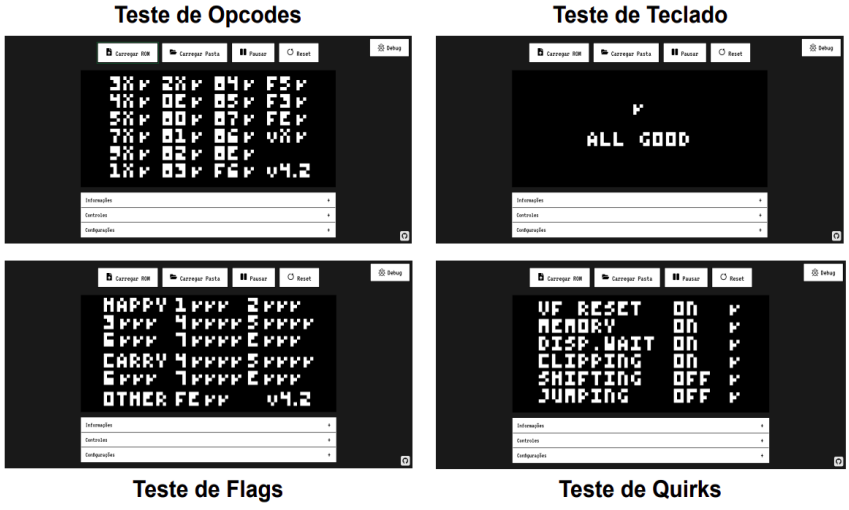
Para assegurar o funcionamento correto do emulador, utilizaram-se ROMs de teste projetadas para verificar o comportamento esperado de cada instrução. Um

exemplo amplamente utilizado é o *CHIP-8 Test Suite*, disponível no GitHub (TIMENDUS, 2021). Esses programas permitem identificar erros de interpretação de *opcodes* e validar a fidelidade da emulação em relação às especificações originais.

O emulador foi submetido a uma bateria completa de testes, abrangendo diferentes aspectos do sistema. Os testes de *opcodes* verificaram se cada instrução era executada corretamente, garantindo que registradores, memória e *stack* fossem manipulados conforme esperado. Os testes de *quirks* avaliaram comportamentos peculiares ou não padronizados do CHIP-8, resultantes de interpretações históricas do sistema, como colisões gráficas e sobrescrita de registradores. Já os testes do teclado confirmaram se a entrada do usuário era reconhecida corretamente, verificando o mapeamento das 16 teclas hexadecimais e a implementação correta da instrução FX0A. Os testes do display validaram a renderização correta de sprites na tela, a detecção de colisões e a atualização precisa do *framebuffer*. Por fim, os testes de temporizadores asseguraram que o *Delay Timer* e o *Sound Timer* decrementassem corretamente a 60 Hz e que os eventos dependentes do tempo fossem disparados de forma precisa.

A Figura 16 apresenta exemplos de alguns dos testes aplicados ao emulador, ilustrando a cobertura abrangente da validação realizada.

Figura 16 – Execução de ROMs de teste



Fonte: Elaborado pelo Autor.

Todos os testes foram aprovados com sucesso, confirmando a precisão, confiabilidade e compatibilidade da implementação com o comportamento esperado do CHIP-8.

4.3 DOCUMENTAÇÃO E VALOR DIDÁTICO

Conforme os objetivos específicos do projeto, o desenvolvimento incluiu a criação de uma documentação técnica e tutorial explicativo da implementação. Essa documentação detalha os conceitos de emulação e arquitetura de computadores, servindo como um guia abrangente e reforçando o caráter didático e acessível do projeto para desenvolvedores e iniciantes.

5 RESULTADOS

Neste capítulo será descrito como o emulador está funcionando atualmente, bem como as etapas necessárias para carregar, executar e testar os programas do sistema CHIP-8. Também será apresentado o funcionamento da interface gráfica, destacando como o usuário pode interagir com os controles e visualizar o estado interno da emulação.

5.1 UTILIZANDO O EMULADOR

A interface gráfica do emulador foi desenvolvida com foco no minimalismo e na facilidade de uso, adotando uma estética de alto contraste, com fundo escuro e fonte monoespaçada, remetendo aos antigos terminais de computação retrô. A Figura 17 ilustra a tela inicial do emulador.

Figura 17 – Tela inicial do Emulador

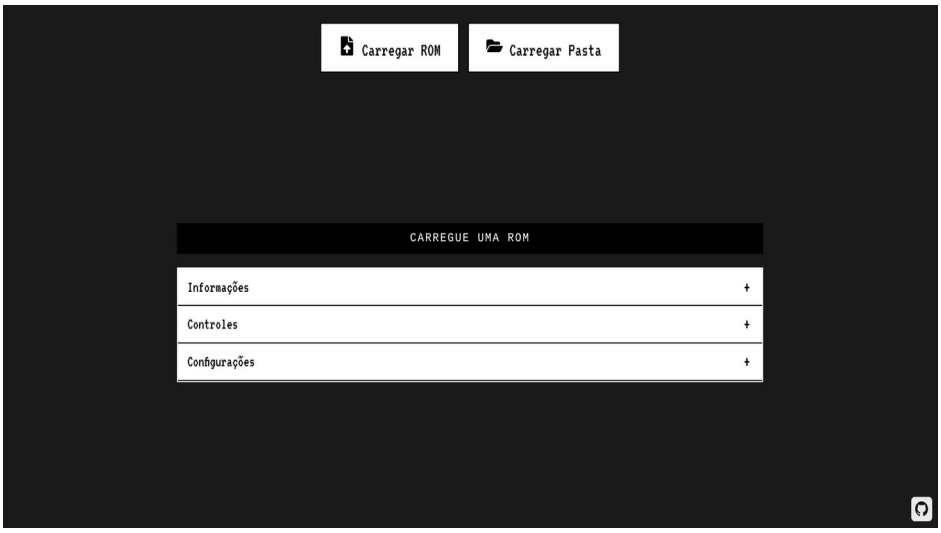
Fonte: Elaborado pelo Autor.

Após a tela de inicialização, o usuário é direcionado para a interface principal do emulador, ilustrada na Figura 18. Nessa tela, encontram-se os botões “Carregar ROM” e “Carregar Pasta”, que permitem respectivamente abrir um arquivo individual no formato .ch8 ou selecionar uma pasta contendo múltiplas ROMs, facilitando o acesso rápido a diferentes programas sem a necessidade de carregá-los um por um.

Na parte inferior da interface, há a seção “Informações”, onde são apresentadas instruções básicas de uso, como o formato aceito de arquivos e observações sobre a reprodução de áudio no navegador, caso o usuário não esteja ouvindo os sons do jogo. A composição dessa tela foi projetada para ser direta, clara e funcional, priorizando a legibilidade e a acessibilidade.

No canto inferior direito, encontra-se também um link para o repositório do projeto no GitHub, permitindo o acesso ao código-fonte e à documentação completa do emulador. A Figura 18 exemplifica a disposição dos elementos da interface de carregamento de ROMs.

Figura 18 – Tela de carregamento de ROMs



Fonte: Elaborado pelo Autor.

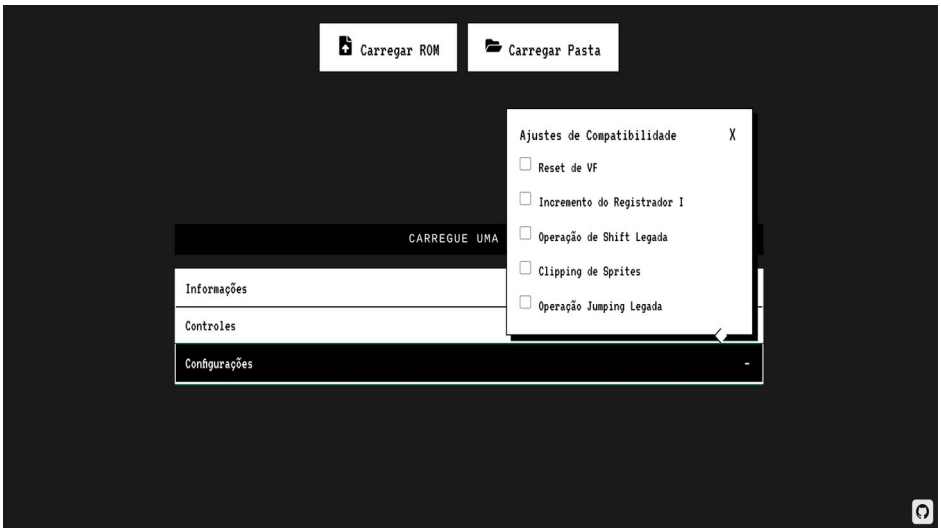
Após o carregamento da ROM, o emulador entra no modo de execução, conforme ilustrado na Figura 20. Nessa etapa, a área central da tela passa a exibir o display do CHIP-8, onde os gráficos do jogo são renderizados em preto e branco, mantendo a estética original do sistema.

Na parte superior, permanecem os botões de controle, acompanhados agora das opções “Pausar”, “Reset” e “Debug”. O botão “Pausar” permite interromper temporariamente a execução da ROM, enquanto o “Reset” reinicia o programa, retornando-o ao estado inicial. Já o botão “Debug” ativa o modo de depuração passo a passo, permitindo que o usuário avance uma instrução por vez. Essa funcionalidade é especialmente útil para fins didáticos e de análise, pois possibilita

observar o comportamento interno do emulador a cada ciclo de execução. A opção “Informações” fornece orientações de uso e instruções práticas, como formatos de ROM compatíveis.

A opção “Configuração” permite ativar ou desativar os *quirks* do CHIP-8. *Quirks* são comportamentos peculiares ou diferenças de implementação que podem afetar o funcionamento de certas instruções. A configuração seletiva desses *quirks* foi necessária, pois alguns deles precisam permanecer desativados para que as ROMs de teste sejam executadas corretamente, garantindo que suas diferentes partes pudessem ser testadas e verificadas de acordo com o comportamento esperado do CHIP-8. No entanto, para a reprodução mais fiel possível, eles podem ser totalmente ativados. A Figura 19 apresenta a tela de Configuração do emulador, na qual o usuário pode ativar ou desativar os *quirks* do CHIP-8.

Figura 19 – Tela de Configuração

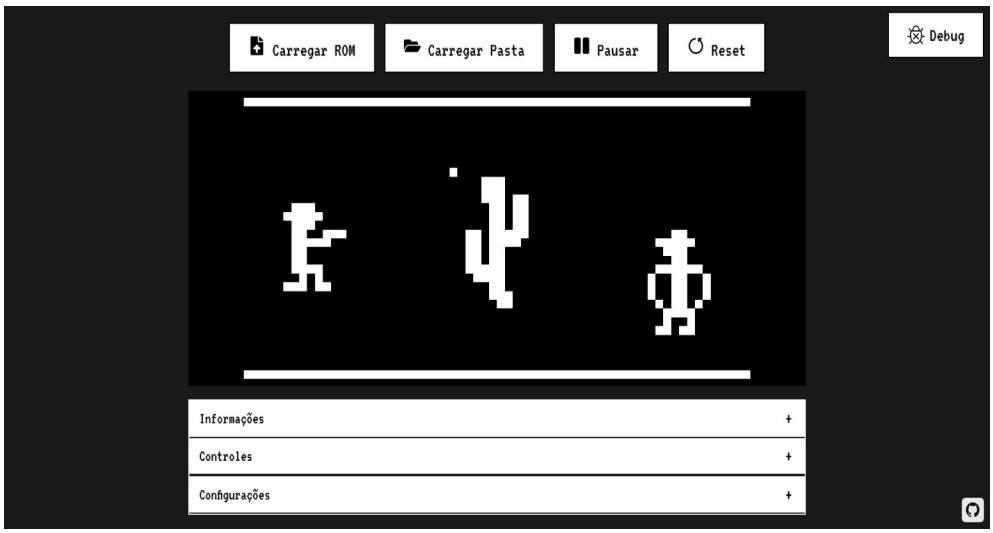


Fonte: Elaborado pelo Autor.

Para obter jogos compatíveis com o CHIP-8, pode-se recorrer a pacotes de ROMs liberados em domínio público. Por exemplo, o “Chip-8 Games Pack” disponível no site Zophar’s Domain reúne títulos como Pong, Tetris, Invaders entre outros (DOMAIN, 2025). Além disso, repositórios no GitHub, como kripod/chip8-roms (KRIPOD, 2023) e mattmikolay/chip-8 (MATTMIKOLAY, 2023), também disponibilizam coleções de ROMs compatíveis com o CHIP-8. Estes arquivos são ideais para testar o funcionamento do emulador com segurança de licenciamento. A

Figura 20 ilustra a execução do jogo Outlaw no emulador, mostrando sua interface e funcionamento.

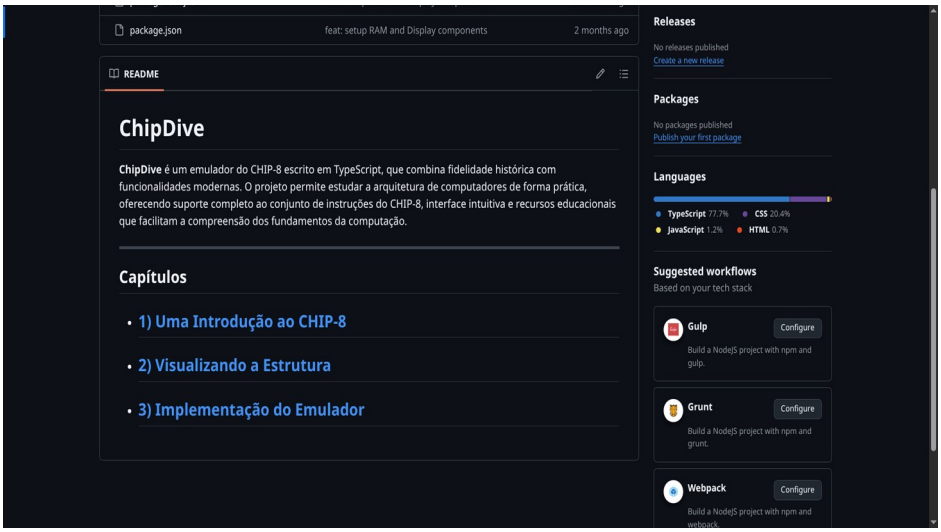
Figura 20 – Execução do jogo Outlaw no emulador



Fonte: Elaborado pelo Autor.

Um dos principais focos do desenvolvimento do projeto foi a documentação detalhada de todas as etapas no GitHub, permitindo que o processo fosse acompanhado e reproduzido por outros desenvolvedores ou estudantes interessados. Essa abordagem teve um caráter didático, priorizando a clareza na explicação das decisões técnicas, estrutura do código e funcionamento do emulador, tornando o repositório não apenas um local de versionamento, mas também uma fonte de aprendizado. A Figura 21 ilustra o repositório do projeto no Github, onde todo o processo foi registrado.

Figura 21 – Documentação do sistema no repositório Github



Fonte: Elaborado pelo Autor.

6 CONCLUSÃO

O desenvolvimento do emulador CHIP-8 alcançou com êxito os objetivos propostos, demonstrando na prática os fundamentos da arquitetura de computadores e da engenharia de *software*. A implementação funcional do sistema comprova a viabilidade técnica do projeto e evidencia seu potencial como uma ferramenta didática eficaz para o ensino de conceitos de baixo nível de computação.

Ao longo do desenvolvimento, foram aplicados princípios sólidos de organização modular, controle de fluxo, manipulação de memória e representação de instruções, permitindo uma compreensão mais profunda do funcionamento interno de um processador. A interface do emulador, projetada para ser intuitiva e acessível, facilitou a interação com o usuário e reforçou o caráter educacional da aplicação.

Além de cumprir seu papel como recurso de aprendizado, o projeto também contribui para a preservação digital da história da computação, mantendo viva uma arquitetura clássica que influenciou gerações de sistemas posteriores. O CHIP-8, reinterpretado por meio deste emulador, não apenas resgata um marco histórico, mas também inspira novas explorações tecnológicas baseadas em simplicidade e clareza estrutural.

Em síntese, o trabalho atingiu plenamente suas metas: o emulador está operacional, bem documentado e apto a servir como ferramenta de apoio ao ensino e à pesquisa. A experiência adquirida durante o desenvolvimento reforça a importância de projetos práticos no aprendizado de sistemas computacionais, incentivando a continuidade de estudos voltados à emulação, arquitetura de processadores e desenvolvimento de ferramentas educacionais interativas.

7 TRABALHOS FUTUROS

Este projeto representa apenas o primeiro passo para explorar o campo da emulação didática e da preservação digital de arquiteturas retro. Ainda há diversas possibilidades de expansão, tanto no aspecto técnico quanto educacional. A seguir, apresentam-se algumas propostas de continuidade:

- **Suporte a Variantes da Arquitetura (SCHIP e XO-CHIP):** adicionar compatibilidade com as versões SUPER-CHIP e XO-CHIP, que introduzem modos gráficos avançados, maior memória e suporte aprimorado a áudio, ampliando o conjunto de ROMs executáveis.
- **Ferramentas Didáticas Avançadas:** expandir o modo *Debug* com visualizações gráficas do ciclo *fetch–decode–execute* e monitores dinâmicos de memória e registradores, tornando o aprendizado mais interativo.
- **Modo de Análise de Performance:** adicionar métricas que exibam em tempo real o consumo de instruções por segundo, ciclos de CPU e eficiência de execução, permitindo estudos sobre otimização.
- **Sistema de Salvamento de Estado (Save States):** permitir pausar, salvar e restaurar a execução da CPU, facilitando a depuração e experimentação de jogos ou programas didáticos.

REFERÊNCIAS

8924TH. Comentário sobre “How to get operation from the instruction (CHIP-8)”. 2023. Disponível em: https://www.reddit.com/r/EmuDev/comments/17fcsqe/how_to_get_operation_from_the_instruction_chip_8/. **Reddit**. Acesso em: 6 nov. 2025.

ATARIWIKI. **CHIP-8**. 2024. Disponível em: https://atariwiki.org/wiki/Wiki.jsp?page=CHIP-8. Acesso em: 7 nov. 2025.

AVENGA. Emulator Vs. Simulator: A Guide For Software Testing. **Avenga**. 2022. Disponível em: https://www.avenga.com/magazine/emulator-vs-simulator/. Acesso em: 17 jan. 2025.

CHIP-8.COM. **Build Your Own CHIP-8 Computer**. 2018. Disponível em: https://chip-8.com/build-your-own-computer. Acesso em: 5 nov. 2025.

CIFALDI, F. **It’s Just Emulation: The Challenge of Selling Old Games**. 2016. Apresentação na Game Developers Conference (GDC). Disponível em: https://www.gdcvault.com/play/1023470/-It-s-Just-Emulation. Acesso em: 8 dez. 2025.

DALE, N.; LEWIS, J. **Ciência da Computação**. 4. ed. Rio de Janeiro: LTC, 2018. ISBN 978-8521635208.

DOMAIN, Z. Chip-8 Games Pack — A pack of simple yet amusing games, in the public domain, for any Chip-8 interpreter. **Zophar’s Domain.** 2025. Disponível em: https://www.zophar.net/pdroms/chip8/chip-8-games-pack.html. Acesso em: 10 out. 2025.

EARNEST, J. **Octo**. 2013. Disponível em: https://github.com/JohnEarnest/Octo. Acesso em: 1 fev. 2025.

EARNEST, J. **CHIP-8 Archive**. 2023. Disponível em: https://johnearnest.github.io/chip8Archive/. Acesso em: 23 jan. 2025.

EMULATION GENERAL WIKI. Emulation accuracy. **Emulation General Wiki**. 2024. Disponível em: https://emulation.gametechwiki.com/index.php/Emulation_accuracy. Acesso em: 23 nov. 2024.

EMULATION GENERAL WIKI. CHIP-8 interpreters. **Emulation General Wiki**. 2025. Disponível em: https://emulation.gametechwiki.com/index.php/CHIP-8_interpreters. Acesso em: 10 jan. 2025.

GEEKSFORGEEKS. What is Memory Management Unit(MMU)?. **GeeksforGeeks**. 2024. Disponível em: https://www.geeksforgeeks.org/what-is-memory-management-unit/. Acesso em: 13 out. 2024.

GLEN. Emulation vs. Porting. **Two Button Crew**, 2020. Disponível em: https://twobuttoncrew.com/2020/10/04/emulation-vs-porting/. Acesso em: 6 nov.

2025.

GREENE, T. P. **Cowgod's Chip-8 Technical Reference**. 1997. Disponível em: http://devernay.free.fr/hacks/chip8/C8TECH10.HTM. Acesso em: 10 jan. 2025.

GULRAK. **CHIP-8 Opcodes Reference**. 2023. Disponível em: https://chip8.gulrak.net/reference/opcodes/. Acesso em: 12 out. 2025.

IFURY25. What's the difference between HLE and LLE?. **Reddit**, 2021. Disponível em: https://www.reddit.com/r/emulation/comments/nfz8fo/whats_the_difference_between_hle_and_lle/. Acesso em: 10 jan. 2025.

INSTRUCTABLES. Using Falstad's Circuit Simulator : 5 Steps. **Instructables,** 2009. Disponível em: https://www.instructables.com/Using-Falstads-Circuit-Simulator/. Acesso em: 3 jan. 2025.

KRIPOD. **chip8-roms**. 2023. Disponível em: https://github.com/kripod/chip8-roms/tree/master. Acesso em: 4 nov. 2025.

LANGHOFF, T. V. Guide to making a CHIP-8 emulator. **Tobias V.Langhoff Blog**, 2020. Disponível em: https://tobiasvl.github.io/blog/write-a-chip-8-emulator/. Acesso em: 15 jan. 2025.

LENOVO. What is a CPU | CPU Types and Functions. **Lenovo US**, 2023. Disponível em: https://www.lenovo.com/us/en/glossary/what-is-cpu/. Acesso em: 15 jan. 2025.

MATTMIKOLAY. **CHIP-8**. 2023. Disponível em: https://github.com/mattmikolay/chip-8/tree/master. Acesso em: 4 nov. 2025.

MIKOLAY, M. CHIP-8 Extensions Reference. **mattmikolay/chip-8 Wiki**. 2017. Disponível em: https://github.com/mattmikolay/chip-8/wiki/CHIP%E2%80%90Extensions-Reference. Acesso em: 17 nov. 2024.

NEUMANN, J. von. **First Draft of a Report on the EDVAC**. 1945. Disponível em: https://web.archive.org/web/20130314123032/http://qss.stanford.edu/~godfrey/vonNeumann/vnedvac.pdf. Acesso em: 3 jan. 2025.

OLDCOMPUTERS.NET. **RCA COSMAC VIP**. 2004. Disponível em: http://oldcomputermuseum.com/cosmac_vip.html. Acesso em: 1 fev. 2025.

PRADA, R. O que é Frame Buffer. **TecMundo**. 2009. Disponível em: https://www.tecmundo.com.br/jogos/1421-o-que-e-frame-buffer.htm. Acesso em: 1 fev. 2025.

RASCIA, T. **Chip8.js**. 2021. Disponível em: https://taniarascia.github.io/chip8/. Acesso em: 3 jan. 2025.

ROLINDAW. CHIP-8 Emulator - Hardware resources vs. virtualized (by CHIP-8 interpreter) resources. **Reddit**, 2024. Disponível em:

https://www.reddit.com/r/EmuDev/comments/1dmmqrk/chip8_emulator_hardware_resources_vs_virtualized/. Acesso em: 17 nov. 2024.

ROTHENBERG, J. Ensuring the longevity of digital documents. **Scientific American**, Scientific American, v. 272, n. 1, p. 42–47, 1995.

SANCHEZ, A. LLE vs HLE and their tradeoffs. **AlexAltea Blog**, 2018. Disponível em: https://alexaltea.github.io/blog/posts/2018-04-18-lle-vs-hle/. Acesso em: 13 out. 2024.

SCHULTZ, J. R. How Silent Hill and Crash Bandicoot devs dealt with the PS1's flaws. **Polygon,** 2024. Disponível em: https://www.polygon.com/playstation/24196061/silent-hill-crash-bandicoot-tech-limitations. Acesso em: 17 nov. 2024.

SCOTFORD, L. CHIP-8 on the COSMAC VIP: Drawing Sprites. **Laurence Scotford Blog**. 2020. Disponível em: https://www.laurencescotford.net/2020/07/19/chip-8-on-the-cosmac-vip-drawing-sprites/. Acesso em: 1 fev. 2025.

SCOTFORD, L. CHIP-8 on the COSMAC VIP: Sound. **Laurence Scotford**, 2020. Disponível em: https://www.laurencescotford.net/2020/07/19/chip-8-on-the-cosmac-vip-sound/. Acesso em: 7 nov. 2024.

SMITH, J. E.; NAIR, R. **Virtual Machines: Versatile Platforms for Systems and Processes**. [S.l.]: Morgan Kaufmann Publishers, 2005. ISBN 978-1558609105.

SPASOJEVIC, A. O que é emulação?. **phoenixNAP Glossário de TI**, 2024. Disponível em: https://phoenixnap.pt/gloss%C3%A1rio/o-que-%C3%A9-emula%C3%A7%C3%A3o. Acesso em: 19 nov. 2024.

TANENBAUM, A. S.; AUSTIN, T. **Organização Estruturada de Computadores**. 6. ed. [S.l.]: Pearson Universidades, 2013. ISBN 8581435394.

TEAM, R. What is Emulation?. **Recalbox Blog**, 2023. Disponível em: https://www.recalbox.com/blog/FAQ-EMULATION/. Acesso em: 5 nov. 2025.

TIMENDUS. **CHIP-8 Test Suite**. 2021. Disponível em: https://github.com/Timendus/chip8-test-suite?tab=readme-ov-file. Acesso em: 4 nov. 2025.

VALADARES, O. CHIP-8 Emulation. **O. Valadares' Blog**, 2024. Disponível em: https://otavio.dev/2024/12/08/chip-8-emulation/. Acesso em: 28 jan. 2025.

WEISBECKER, J. An easy programming system. **BYTE**, v. 3, n. 12, p. 108–122, dec 1978. Disponível em: https://archive.org/details/byte-magazine-1978-12. Acesso em: 16 jan. 2025.

WEISBECKER, J. A Microprocessor for the Revolution: The RCA COSMAC VIP. **BYTE**, 1978. Disponível em: https://archive.org/details/byte-magazine-1978-12/page/n109/mode/2up?view=theater. Acesso em: 17 nov. 2024.

WEISBECKER, J. Representação de um sprite de foguete. **BYTE**, 1978. Disponível em: https://archive.org/details/byte-magazine-1978-12/page/n113/mode/2up?view=theater. Acesso em: 17 nov. 2024.

WINTER, D. **David Winter's CHIP-8 Emulation Page**. 1999. Disponível em: https://www.pong-story.com/chip8/. Acesso em: 5 nov. 2025.

XAPSOS, M. A. et al. **How Long Can the Hubble Space Telescope Operate Reliably? – A Total Dose Perspective**. 2004. NASA Technical Reports Server. Disponível em: https://web.archive.org/web/20170227173247/https://ntrs.nasa.gov/archive/nasa/casi.ntrs.nasa.gov/20160005759.pdf. Acesso em: 17 nov. 2024.