UNIVERSIDADE ESTADUAL PAULISTA "JÚLIO DE MESQUITA FILHO"
FACULDADE DE ENGENHARIA DE ILHA SOLTEIRA

LEONARDO GONÇALVES DE MELO JUNIOR

EXPLORANDO A LINGUAGEM ASSEMBLY X86 NA CRIAÇÃO DE JOGOS INTERATIVOS

LEONARDO GONÇALVES DE MELO JUNIOR

EXPLORANDO A LINGUAGEM ASSEMBLY X86 NA CRIAÇÃO DE JOGOS INTERATIVOS

Trabalho de graduação apresentado à Faculdade de Engenharia, Câmpus de Ilha Solteira - Universidade Estadual Paulista "Júlio de Mesquita Filho" como parte dos requisitos para obtenção do título de Bacharel em Engenharia Elétrica.

Orientador: Prof. Dr. Carlos Antonio Alves

ILHA SOLTEIRA
2023

FICHA CATALOGRÁFICA
Desenvolvido pelo Serviço Técnico de Biblioteca e Documentação

Melo Junior, Leonardo Gonçalves de.
M528e Explorando a linguagem Assembly x86 na criação de jogos interativos / Leonardo Gonçalves de Melo Junior. -- Ilha Solteira: [s.n.], 2023
62 f. : il.

Trabalho de conclusão de curso (Graduação em Engenharia Elétrica) - Universidade Estadual Paulista. Faculdade de Engenharia de Ilha Solteira, 2023

Orientador: Carlos Antonio Alves

Inclui bibliografia

1. Linguagem de montagem. 2. Jogos digitais. 3. Programação de jogos. 4. Vscode. 5. Design de jogos.

Amanda Sertori dos Santos
Bibliotecária - CRB/8-9061
Seção Técnica de Referência, Atendimento ao Usuário e Documentação
Diretoria Técnica de Biblioteca e Documentação

UNIVERSIDADE ESTADUAL PAULISTA
"JÚLIO DE MESQUITA FILHO"
Câmpus de Ilha Solteira

ATA DE DEFESA DE TRABALHO DE GRADUAÇÃO

Aos vinte e um dias do mês de dezembro do ano de dois mil e vinte e três, o discente ***Leonardo Gonçalves de Melo Junior,*** matriculado sob o nº 161052274, tendo como banca examinadora o seu orientador, *o Prof. Dr. Carlos Antonio Alves, o Prof. Dr. Alexandre Cesar Rodrigues da Silva e o Doutor Lucas do Carmo Yamaguti,* apresentou o Trabalho de Graduação intitulado **"Explorando as capacidades da linguagem assembly x86 na criação de jogos interativos",** obtendo a nota 9.0 (NOVE) e conceito APROVADO.

Prof. Dr. Carlos Antonio Alves
- Orientador-

Leonardo Gonçalves de Melo Junior
- Discente -

Prof. Dr. Alexandre Cesar Rodrigues da Silva
- Membro da Banca -

Doutor Lucas do Carmo Yamaguti
- Membro da Banca -

Faculdade de Engenharia – Departamento de Engenharia Elétrica
Avenida Professor José Carlos Rossi, 1370 - CEP 15.385-000, Ilha Solteira - SP - Brasil
Tel (18) 37431150 - Fax (18) 3742-2735 - www.feis.unesp.br

AGRADECIMENTOS

Primeiramente gostaria de agradecer à minha família, por sempre se esforçarem para me dar apoio, independente do momento, sempre presentes e vibrando em minhas conquistas e me ensinando, através do exemplo, sobre humildade, responsabilidade e afeto.

Agradeço à minha companheira, Elis, por existir no mundo e me permitir fazer parte de sua vida, sua parceria foi ponto chave em toda jornada deste trabalho, obrigado.

Agradeço imensamente quem me ajudou neste projeto, em primeiro lugar meu orientador Carlos Alves, que aceitou me orientar neste grande desafio. Também agradeço a Kyle Vassau, por ser extremamente solícito ao me responder sobre seu trabalho com a confecção de seu jogo.

Também agradeço às pessoas que de alguma forma se fizeram presentes durante a graduação em Ilha Solteira, principalmente meus amigos de república e da turma 2016/01, alguns dos quais irei levar para toda minha vida.

Por último, agradeço aos amigos que fiz durante minha vida, meus amigos de infância da “mesa redonda”, dos trabalhos que tive, das escolas que estudei em Iturama e Araçatuba, mesmo que tenha perdido o contato de alguns, o carinho se mantém.

RESUMO

Este trabalho de graduação foi motivado pelo interesse na relevância e aplicabilidade da linguagem de montagem, especialmente no contexto de jogos digitais. Ao longo da graduação, o ensino da linguagem de montagem revelou-se um campo muito interessante de estudo e, com este trabalho, foi possível verificar a importância técnica e didática da linguagem de montagem no tema de circuitos integrados e aplicação em programação de jogos digitais. Este trabalho começou com uma análise histórica, foram discutidos os jogos desde suas origens até os digitais modernos, foi ressaltado seu impacto cultural e social e abordada a evolução tecnológica associada. O estudo também contemplou o panorama da indústria de jogos, analisou tanto o mercado quanto avanços tecnológicos, incluiu previsões de crescimento de mercado consumidor e de manufatura de microchips. Foi abordado o uso da linguagem de montagem em jogos digitais, que culminou no desenvolvimento de um jogo, no qual foi demonstrado experimentalmente como os conceitos teóricos podem ser aplicados à programação em linguagem de montagem e ao design de jogos. Este estudo destaca as vantagens da linguagem de montagem, como eficiência e controle direto do hardware, mas também explicita suas desvantagens, como a complexidade e a dificuldade de compatibilidade com outras arquiteturas, as características discutidas foram exemplificadas pela criação experimental de elementos de um jogo digital em ambiente de simulação compatível com a época do lançamento da arquitetura 8086.

Palavras-chave – Linguagem de montagem. Jogos Digitais. Programação de jogos. Vscode. NASM. *Design* de jogos.

ABSTRACT

This academic paper was motivated by an interest in the relevance and applicability of assembly language, especially in the context of digital games. Throughout the course, the teaching of assembly language proved to be a very interesting field of study, and with this work, it was possible to verify the technical and didactic importance of assembly language in the theme of integrated circuits and application in digital game programming. This work began with a historical analysis, discussing games from their origins to modern digital ones, emphasizing their cultural and social impact and addressing the associated technological evolution. The study also contemplated the overview of the game industry, analyzing both the market and technological advances, including forecast of consumer market growth and microchip manufacturing. The use of assembly language in digital games was explored, culminating in the development of a game, in which it was experimentally demonstrated how the theoretical concepts can be applied to programming in assembly language and game design. This study highlights the advantages of assembly language, such as efficiency and direct hardware control, but also explicitly states its disadvantages, such as complexity and difficulty of compatibility with other architectures, the characteristics discussed were exemplified by the experimental creation of elements of a digital game in a simulation environment compatible with the time of the launch of the 8086 architecture.

Keywords – Assembly Language. Digital Games. Game Programming. Vscode. NASM. Game design.

LISTA DE FIGURAS

LISTA DE SIGLAS E ABREVIATURAS

ARM	*Advanced* RISC *Machine* (originalmente *Acorn* RISC *Machine*)
AT&T	*American Telephone & Telegraph Company*
BIOS	*Basic Input/Output System*
CCT	Celular, computador ou *tablet*
CISC	*Complex Instruction Set Computer*
CPU	*Central Processing Unit*
DEC	*Digital Equipment Corporation*
EDVAC	*Electronic Discrete Variable Automatic Computer*
EUV	*Extreme Ultraviolet*
GPU	*Graphics Processing Unit*
IAS	*Institute for Advanced Studies*
IBM	*International Business Machines*
LIFO	*Last In, First Out*
LSI	*Large Scale Integration*
MIT	*Massachusetts Institute of Technology*
MSI	*Medium Scale Integration*
PC	*Personal Computer*
PDP-1	*Programmed Data Processor-1*
QEMU	*Quick Emulator*
RISC	*Reduced Instruction Set Computer*
ROM	*Read-Only Memory*
SAIL	*Self-Aligned Imprint Lithography*
SARS-CoV-2	*Severe Acute Respiratory Syndrome Coronavirus 2*
SSI	*Small Scale Integration*
TR-55	*Transistor Radio - 1955*
TSMC	*Taiwan Semiconductor Manufacturing Company*
TV	Televisão
UFMG	Universidade Federal de Minas Gerais
UC	Unidade de Controle

ULA	Unidade Lógica e Aritmética
ULSI	*Ultra Large-Scale Integration*
VGA	*Video Graphics Array*
VLSI	*Very Large-Scale Integration*

SUMÁRIO

1 INTRODUÇÃO

Este Trabalho de Graduação buscou mostrar dados sobre o mercado e a indústria de jogos digitais ao fazer uma análise histórica sobre o avanço em tecnologias de hardware e software, o estudo foi finalizado na construção de um jogo em linguagem de montagem, referência ao surgimento dos primeiros jogos digitais, que precisavam utilizar a capacidade máxima do hardware em que eram executados. No capítulo 1 foi abordada a linha do tempo dos jogos e as definições do que são jogos, segundo estudiosos da área. O próximo capítulo abordou o mercado e perspectiva da indústria de jogos para diferentes plataformas ao longo dos últimos anos, houve ênfase no setor de jogos educacionais e no impacto da pandemia na indústria de jogos. No terceiro capítulo, foi analisado o avanço tecnológico na indústria de manufatura de circuitos integrados, desde os primeiros computadores até os atuais. Estudada a manufatura, o próximo capítulo considerou a linguagem de montagem e sua importância como parte intrínseca do processo de criação de hardware e softwares, com atenção especial à arquitetura de John Von Neumann. Posterior ao desenvolvimento da linguagem de montagem, o quarto capítulo também explora a aplicabilidade da linguagem de montagem em jogos digitais, culminando no quinto capítulo em que foi explicado o processo de montagem do jogo e captura de tela ilustrando seu funcionamento.

1.1 A HISTÓRIA DOS JOGOS E SEU CONCEITO

A história dos jogos na sociedade remonta a milhares de anos e reflete a necessidade humana de competir, desafiar a mente e buscar entretenimento. Ao longo do tempo, os jogos em diversas formas desempenharam um papel fundamental na evolução da cultura e da sociedade, além de refleti-las.

1.2 AS FUNÇÕES DOS JOGOS DESDE SEUS PRIMÓRDIOS

Desde os primórdios da civilização foi possível encontrar alguma forma de interação voluntária com regras pré-definidas. Jogos tornaram-se uma parte intrínseca da experiência humana, o que proporcionou não só diversão, mas também oportunidades de interação social e desenvolvimento intelectual e estratégico.

No Brasil, antes da chegada dos portugueses, havia jogos praticados pelos indígenas, por exemplo a peteca (SANTOS, 2020), e, no mundo, alguns dos jogos mais antigos que se

tem registro são jogos de tabuleiro, milenares, como o “Jogo Real de Ur”, o “Mancala” e “Senet”, (ALFIUENE; CUSTÓDIO, 2019) eles demonstraram como os jogos foram uma parte importante e duradoura da história humana, moldando a maneira como interagimos uns com os outros (HUIZINGA, 2014:1971).

Figura 1 - Jogo Real de Ur.

Fonte: (PARK, 2021).

Em relação à preservação e disseminação de culturas ao longo dos tempos, os jogos tiveram sua parcela de importância, sendo frequentemente usados para transmitir conhecimento e tradições. Essa integração de história e entretenimento evidencia a capacidade dos jogos de fazer pontes entre o passado e o presente, fornecendo um meio para que as pessoas se conectem com suas raízes culturais (MIRANDA, 2002).

Desde os primórdios dos jogos de tabuleiro, até os mais recentes avanços tecnológicos, os jogos são parte fundamental da experiência humana, que em seu núcleo parte de simples elementos que independem de altas tecnologias para serem aplicáveis. Jogos analógicos continuam presentes em nossa cultura, e mesmo com o avanço tecnológico na área de jogos digitais, jogos de tabuleiro modernos como “Brass: Birmingham (2018)” ou “Ark Nova (2021)” (BOARDGAMEGEEK, 2023) atraem bases de fãs dedicadas, e evidenciam a durabilidade do design de jogos (MIRANDA, 2002).

Por meio do estudo destes jogos históricos é possível deduzir a manutenção e evolução das atividades recreativas como um componente consistente da evolução humana. A história

dos jogos é uma narrativa de diversidade e inovação e uma homenagem à busca constante por diversão, desafio e conexão (TEKINBAS; ZIMMERMAN, 2003).

À medida que investigamos as várias formas de jogos, desde os jogos mais antigos que se tem conhecimento até as mais recentes criações digitais, não estamos apenas explorando a diversão e o entretenimento que lhes estão associados, mas também a cultura e a tradição que foi, e continua a ser, incorporada. Esta extensa herança é a base na qual os jogos modernos são construídos e teve um impacto significativo na indústria de jogos como um todo, desempenhando um papel significativo na transformação da indústria de jogos em um setor cultural e econômico de importância global (TEKINBAS; ZIMMERMAN, 2003).

As interações e dinâmicas sociais do mundo real são levadas para o ambiente do jogo, portanto relações, hierarquias, amizades, rivalidades e normas sociais que existem fora do jogo podem influenciar a forma como os jogadores se comportam, interagem e se relacionam dentro do jogo. Isso é especialmente relevante em jogos *multiplayer* ou sociais, onde as interações entre jogadores podem ser afetadas por fatores externos, como amizades, competições ou mesmo conflitos do mundo real (TEKINBAS; ZIMMERMAN, 2003).

Quando os jogadores desviam das regras estabelecidas em um jogo, eles começam a moldar a experiência do jogo de maneira única, como *designers*, tomando decisões criativas que podem afetar o curso do jogo. Eles passam de meros participantes para influenciadores ativos na experiência do jogo (TEKINBAS; ZIMMERMAN, 2003).

Os jogos transcendem as fronteiras da mera diversão. O "Banco Imobiliário" não é meramente um passatempo divertido, mas um documento cultural que espelha características de nosso sistema econômico. As cartas de baralho, com origens na França, não se limitam a letras e desenhos, elas representam uma interação cultural rica, incorporando simbologias que ecoam uma realidade ideológica mais ampla. O xadrez, não é apenas um desafio estratégico, mas também uma representação simbólica de batalhas históricas, refletindo as complexas interações entre culturas e sociedades (TEKINBAS; ZIMMERMAN, 2003).

Os jogos, todos eles, são entrelaçados com a cultura, e, da mesma forma que podem ser analisados por suas qualidades formais e experiências proporcionadas, também devem ser interpretados como objetos culturais em si (TEKINBAS; ZIMMERMAN, 2003).

Investigar a história dos jogos pode nos dar uma compreensão mais profunda de como os jogos, independentemente da sua forma, tiveram uma grande influência na cultura ao longo dos tempos, trazendo um rico legado de entretenimento que não só diverte, mas também ensina lições sobre estratégia, interação social e inovação.

A história dos jogos é uma demonstração contínua de criatividade, competição e conexão humana, e é um contexto essencial para a compreensão da evolução dos jogos digitais, que se tornaram uma indústria muito importante na sociedade moderna.

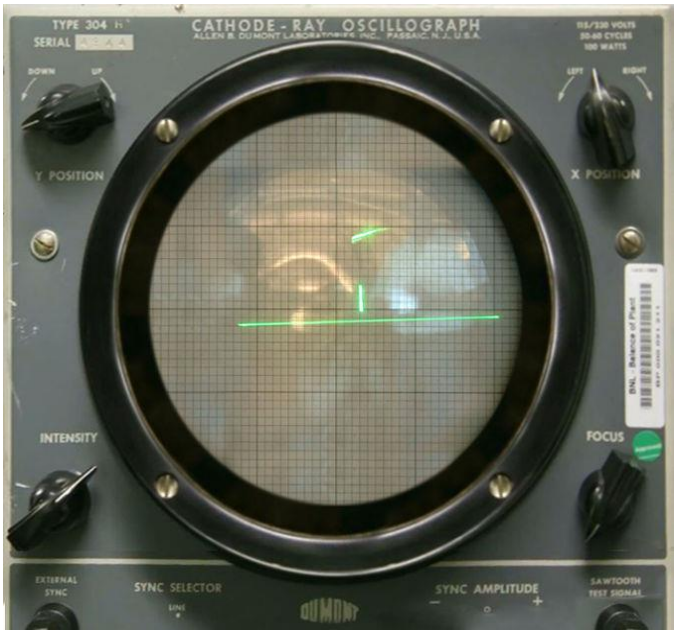
1.3 SURGIMENTO DOS JOGOS DIGITAIS

A evolução dos jogos não apenas refletiu as mudanças culturais humanas, mas também foi acompanhada de perto pelos avanços tecnológicos das suas respectivas épocas. À medida que a tecnologia avançava, os jogos também, tornando-se mais complexos e imersivos, derivando diretamente do contexto histórico e da tecnologia disponível.

No entanto, a revolução que mais diretamente afetou a comunidade de jogos ocorreu na segunda metade do século XX, envolvendo empresas como IBM (*International Business Machines*), Apple e Microsoft, quando os computadores se tornaram mais comuns. Esta era, principalmente nas décadas de 1970 e 1980, foi caracterizada como a "corrida dos computadores", em que cientistas e engenheiros tentaram criar as máquinas mais poderosas e inovadoras, buscando o domínio de um mercado em ascensão (HERMAN, 2001).

Anterior a este contexto temos exemplos tidos como as primeiras existências de jogos digitais como, por exemplo, o *"Tennis for Two"*, criado por Willy Higinbotham, em 1958, físico do Laboratório Nacional de Brookhaven. O jogo foi implementado em um osciloscópio, como ilustra a Figura 2 (HERMAN, 2001).

Figura 2 - *"Tennis for Two"* em um osciloscópio DuMont Lab tipo $304-A$.



Fonte: (AFTERLIFE, 2016)

“Tennis for Two” foi um sistema de entretenimento eletrônico que, ao utilizar um osciloscópio e duas pequenas caixas com um botão e uma alavanca, em cada, permitiu que os visitantes do Laboratório Nacional de *Brookhaven* interagissem com um jogo simples de tênis em uma tela (HERMAN, 2001).

Embora *"Tennis for Two"* tenha sido um marco na história dos jogos eletrônicos, ele não foi patenteado, o que permitiu que outros visionários explorassem esse novo meio de entretenimento. Assim, durante a década de 1970, no contexto da “corrida dos computadores” vários sistemas de videogames começaram a surgir, antes mesmo do lançamento do famoso Atari 2600. Esses sistemas incluíam o *Magnavox Odyssey*, um dos primeiros consoles domésticos, e o pioneiro Pong, criado pela Atari, que se tornou um grande sucesso nos fliperamas (HERMAN, 2001).

No entanto, o lançamento do Atari 2600 em 1977 revolucionou o mundo dos jogos. Com um amplo número de jogos e controles intuitivos, o Atari 2600 se tornou o líder da indústria de videogames na época. Esse lançamento desencadeou uma avalanche de desenvolvimento e inovação na área de jogos eletrônicos, e seu legado persiste até hoje (HERMAN, 2001).

A crescente popularidade dos jogos eletrônicos coincidiu com o desenvolvimento dos primeiros computadores pessoais. Esses primeiros computadores compartilhavam algumas semelhanças com os consoles de videogame da época, uma vez que muitos deles eram projetados para executar programas de entretenimento, além de tarefas mais funcionais. Foi nesse contexto que surgiu o "Spacewar! ", mostrado na Figura 3, amplamente considerado o primeiro jogo de computador.

Desenvolvido em 1962 no MIT (*Massachusetts Institute of Technology*), *“Spacewar!”* colocou os jogadores no comando de naves e envolveu batalhas no espaço. O jogo foi projetado para ser executado em um computador DEC (*Digital Equipment Corporation*), PDP-1 (*Programmed Data Processor-1*) o primeiro minicomputador produzido, lançado em 1960, que possuía um sistema com arquitetura pioneira e inovações tecnológicas, utilizava uma arquitetura de $18\,bits$, com $4\,Kbytes$ de memória, com execução por meio de cartões perfurados.

Esses avanços na tecnologia da computação abriram caminho para novas dimensões nos jogos, permitindo aos desenvolvedores explorar conceitos mais complexos e gráficos aprimorados. *“Space War!”* não apenas definiu a base dos jogos de computador, mas também demonstrou o potencial ilimitado de entretenimento que poderia ser alcançado à medida que a

tecnologia continuasse a evoluir. Assim, a combinação de *videogames* e computadores moldou o cenário dos jogos como o conhecemos hoje.

Figura 3 - *"Spacewar!"* no Museu da História do Computador, em 2007.

Fonte: (REDAÇÃO, 2016).

Para compreender melhor a popularidade e crescimento dos jogos, em conjunto com o avanço do hardware, é necessário levar em conta o que define um jogo, e como essa definição se aplica aos jogos digitais, que surgiram por volta de 65 anos atrás e hoje fazem parte de um mercado enorme que vem expandindo muito nos últimos anos.

1.4 ELEMENTOS NECESSÁRIOS NO DESENVOLVIMENTO DE UM JOGO

Há um vasto campo de estudo sobre jogos, do qual diversos conceitos e definições foram propostos, refletindo a diversidade de visões sobre o que constitui um jogo. Os autores citados neste capítulo, cada um a partir de sua área de especialização, contribuíram para o entendimento da definição de jogos e os elementos presentes no mesmo.

Os conceitos de jogo definidos por Johan Huizinga, Chris Crawford e Jesper Juul são fundamentais para entender a natureza interativa e cultural dos jogos. Johan Huizinga cita (1980, p.28, tradução livre):

> Jogar é uma atividade ou ocupação voluntária realizada dentro de certos limites fixos de tempo e lugar, de acordo com regras livremente aceitas, mas absolutamente

> vinculativas, tendo como objetivo a própria atividade e acompanhada por uma sensação de tensão, alegria e a consciência de que é 'diferente' da 'vida comum'.

Para ele, historiador, teórico e crítico cultural, os jogos antecedem a humanidade e, portanto, não são características exclusivamente nossas, sendo possível verificar essas características em interações de diversos animais. A definição de Huizinga, anterior ao surgimento de jogos digitais, por possuir uma definição ampla, é aplicável em diversos jogos, mesmo digitais, como no FIFA (SPORTS, 1993), onde os jogadores se engajam voluntariamente em uma atividade regida por regras específicas e visam ao sucesso dentro da esfera do jogo, sem propósito prático fora dele.

Chris Crawford, em *"The Art of Computer Game Design"*, cita jogos digitais como uma forma de arte interativa, com o jogador tendo papel central. Para Crawford, a tomada de decisão é um elemento fundamental nos jogos, que são estruturados em torno dela, impactando o desfecho e possibilidades de vitória ou derrota. Análoga à tomada de decisão, existe a resposta do jogo à ação do jogador, criando um diálogo entre jogo e jogador, esta resposta reforça o envolvimento e imersão do jogador (CRAWFORD, 1984).

Para Crawford, o jogo é um espaço seguro onde é possível experimentar decisões e estratégias não tendo consequências no mundo real. Huizinga também aborda a ideia de segurança no contexto de jogos, em que menciona a existência de “círculos mágicos” onde as regras do jogo se aplicam, proporcionando um ambiente separado do mundo real, no qual as pessoas podem se envolver em atividades lúdicas sem as consequências sérias que teriam fora desse espaço (CRAWFORD, 1984) (HUIZINGA, 2014:1971).

Jesper Juul apresenta uma definição que equilibra os aspectos clássicos dos jogos com a realidade das narrativas dos jogos modernos. Juul argumenta que jogos digitais representam uma interseção entre ficção e regras. As regras descrevem o que pode ou não pode ser feito no jogo, propondo assim que durante a atividade, os desafios gerados pelas regras devem ser superados pelos jogadores. A ficção permite aos jogadores imaginar e imergir em mundos ficcionais criados por eles mesmos (JUUL, 2011).

Para Juul, a ficção ajuda o jogador a compreender as regras do jogo, e as regras incentivam o jogador a imergir no mundo ficcional imaginado, portanto é o jogador, sua imaginação e suas ações dentro das regras que trazem o jogo à vida (JUUL, 2011).

Huizinga se concentra na experiência cultural e tradicional dos jogos como atividades separadas da vida real, Crawford traz a interatividade e tomada de decisões e Juul reconhece tanto a estrutura clássica dos jogos quanto a relevância das narrativas e da representação

ficcional. Embora os conceitos dos autores não sejam excludentes, apresentam ênfases diferentes, que refletem seus diferentes campos de estudos.

Cada autor contribui para uma compreensão mais profunda do que são os jogos, sendo mais do que sistemas fechados de regras, possuindo experiências que podem ser profundamente imersivas, interativas e narrativas. A complementação de conceitos é fundamental para concepção de jogos, assim como para seu desenvolvimento, proporcionando uma base sólida de conceitos que podem ser incorporados em elementos de seu próprio design de jogo.

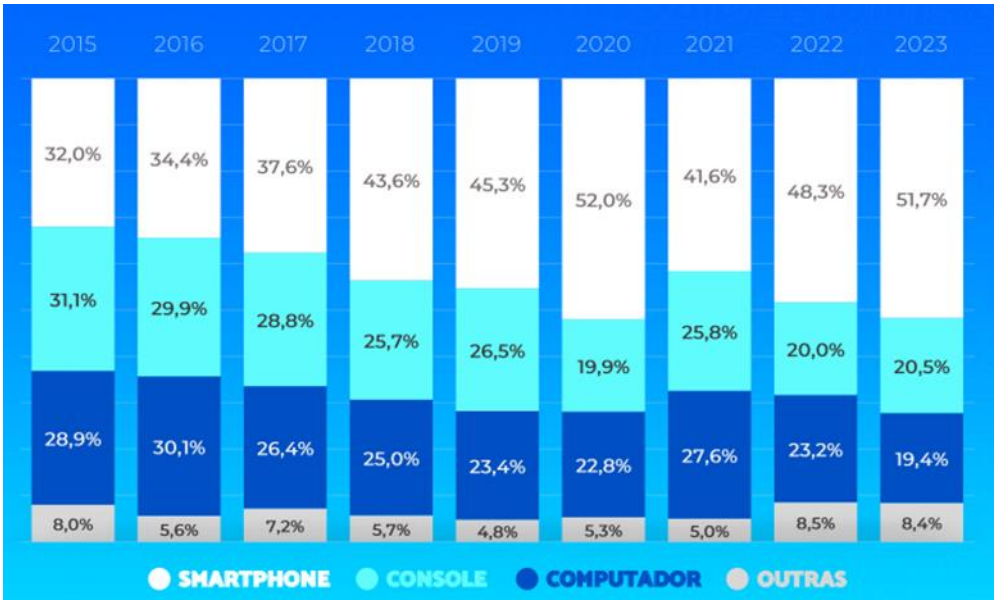
2 A INDÚSTRIA DE JOGOS NO BRASIL: CRESCIMENTO DE MERCADO

A indústria de jogos global, revela uma expansão econômica notável e uma inovação contínua. Espera-se que o mercado global de jogos gere US$ 249,7 bilhões em 2023, com expectativa de crescimento com taxa anual de 9,83% entre 2023 e 2027 (STATISTA, 2023).

2.1 PERSPECTIVA DE CRESCIMENTO DO MERCADO DE JOGOS NO BRASIL

Para o mercado brasileiro, a expectativa de receita em 2023 é de US$ 2,22 bilhões, com o mercado *mobile*, sozinho, sendo responsável por US$ 1,08 bilhões. Como visto na Figura 4, o smartphone teve o maior crescimento ao longo dos últimos anos, se destacando por atingir um público maior e mais diversos, saltando de 32% em 2015, para 51,7% em 2023, como plataforma preferida entre os jogadores (GROUP; GAMERS, 2023).

Figura 4 - Preferência de plataforma para jogar entre os entrevistados brasileiros.



Fonte: (GROUP; GAMERS, 2023).

Com a inserção de jogos no dispositivo eletrônico mais difundido do mundo em 2023, alguns setores, além do setor de entretenimento, dividem espaço nesse mercado.

2.2 CRESCIMENTO DOS JOGOS EDUCACIONAIS E GAMIFICAÇÃO

O setor dos jogos educacionais, têm visto um crescimento acelerado, tanto em valor financeiro quanto em número de usuários, ganhando mais espaço nesse nicho, com receita global de US$ 15,6 bilhões em 2022 (IMARC, 2022). Este segmento se beneficia bastante do investimento em gamificação[1] e do desenvolvimento de novas experiências de aprendizagem buscando maior envolvimento.

Plataformas como *Duolingo*, que utilizam gamificação para engajamento dos usuários, ilustram a competição com outras formas de entretenimento digital. Estas plataformas disputam com redes sociais e jogos de entretenimento o interesse do seu público. Csikszentmihalyi descreve o "Fluxo", um estado de profunda concentração e foco que é definido pelo autor como "um estado em que as pessoas estão tão envolvidas em uma atividade que nada mais parece importar" (CSIKSZENTMIHALYI, 1975). Este estado representa uma experiência ótima caracterizada por um foco profundo e prazer na própria atividade, em vez de quaisquer recompensas externas que ela possa trazer. Foram identificados, pelo autor, nove elementos indicativos do fluxo, que são equilíbrio entre desafio e habilidades, fusão de ação e consciência, metas claras, *feedbacks* claros, concentração na tarefa em mãos, sensação de controle, perda de autoconsciência, transformação da noção de tempo e experiência autotélica. Para o autor, esses elementos contribuem para a profunda imersão e desempenho aprimorado de uma pessoa na tarefa (CSIKSZENTMIHALYI, 1975).

2.3 IMPACTO E TRANSFORMAÇÕES NA INDÚSTRIA DE JOGOS DURANTE A PANDEMIA

A pandemia do coronavírus SARS-CoV-2 (*Severe Acute Respiratory Syndrome Coronavirus 2*) desencadeou uma expansão inesperada na indústria de jogos digitais. Enfrentando um contexto de isolamento e incerteza global, a indústria viu um aumento significativo na demanda, um subproduto das circunstâncias globais. Em 2023, a receita global da indústria de jogos foi de US$ 187,7 bilhões, crescendo 2,6% em relação a 2022.

[1]Gamificação ou, em inglês, *gamification*, é um termo utilizado para se referir ao uso de elementos oriundos de jogos em contextos exteriores a eles, como por exemplo o uso de pontuações em atividades de sala de aula (MURR; FERRARI, 2020).

(NEWZOO, 2023). Esse crescimento foi impulsionado em grande parte pelo avanço dos jogos para dispositivos móveis, que representaram cerca de metade da receita total do mercado.

A adaptação ao ambiente remoto foi uma resposta necessária. Eventos do setor, incluindo torneios de esporte eletrônico, migraram para o ambiente virtual, assegurando a continuidade da interação comunitária e mantendo a comunidade de jogadores engajada. Os jogos digitais emergiram não apenas como uma forma de entretenimento, mas também como um meio de apoio ao bem-estar psicológico, atuando como uma válvula de escape frente à ansiedade social generalizada.

A Pesquisa Game Brasil 2021 apontou que 72% da população do país joga jogos eletrônicos no geral, e 51,5% dos gamers realizaram mais sessões de partidas online com amigos durante a pandemia (NEWZOO, 2021). Plataformas populares como *Steam* viram um aumento recorde no número de usuários ativos diários, com um pico de 23,5 milhões em março de 2020, de acordo com as estatísticas de usuários da *Steam*. Em novembro de 2023 a *Steam* atingiu um pico de 31 milhões de jogadores online, como ilustra a Figura 5 (STEAM, 2023).

Figura 5 – Quantidade máxima de jogadores online, simultaneamente.

Fonte: (STEAM, 2023).

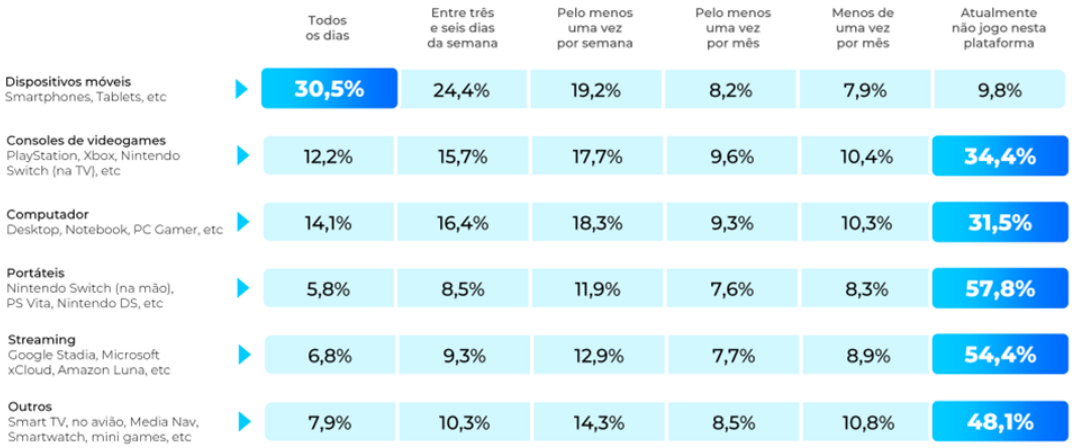
Enquanto os jogos sociais ganhavam força, permitindo conexões em um momento em que o distanciamento físico era essencial, a indústria explorava novos caminhos de inovação. Estes números e tendências, apesar de impressionantes, levantam questões sobre a sustentabilidade deste elevado crescimento no cenário pós-pandêmico.

2.4 CRESCENTE EXPOSIÇÃO ÀS TELAS

A pesquisa feita pelo Programa de Pós-graduação em Saúde Pública da Faculdade de Medicina da UFMG (Universidade Federal de Minas Gerais), expõe o aumento de horas de lazer gastas em CCT (celular, computador ou tablet), para a população adulta, entre 2016 e

2021, em comparação com as horas gastas assistindo TV (Televisão), que seguem estagnadas (CARDOSO; CALDEIRA; SOUSA; CLARO, 2023). O aumento serial do acesso da população às telas, principalmente via smartphones, impulsionou a indústria de jogos em celular que liderou o ranking de frequência de uso, como demonstra a Figura 6.

Figura 6 - Frequência de uso de cada plataforma, para jogar.



	Todos os dias	Entre três e seis dias da semana	Pelo menos uma vez por semana	Pelo menos uma vez por mês	Menos de uma vez por mês	Atualmente não jogo nesta plataforma
Dispositivos móveis Smartphones, Tablets, etc	**30,5%**	24,4%	19,2%	8,2%	7,9%	9,8%
Consoles de videogames PlayStation, Xbox, Nintendo Switch (na TV), etc	12,2%	15,7%	17,7%	9,6%	10,4%	**34,4%**
Computador Desktop, Notebook, PC Gamer, etc	14,1%	16,4%	18,3%	9,3%	10,3%	**31,5%**
Portáteis Nintendo Switch (na mão), PS Vita, Nintendo DS, etc	5,8%	8,5%	11,9%	7,6%	8,3%	**57,8%**
Streaming Google Stadia, Microsoft xCloud, Amazon Luna, etc	6,8%	9,3%	12,9%	7,7%	8,9%	**54,4%**
Outros Smart TV, no avião, Media Nav, Smartwatch, mini games, etc	7,9%	10,3%	14,3%	8,5%	10,8%	**48,1%**

Fonte: (GROUP; GAMERS, 2023).

Como dito anteriormente, a popularização de jogos no dispositivo eletrônico mais difundido do mundo em 2023 teve um impacto muito alto na frequência de uso de *smartphones*, o que também reflete no crescimento da demanda por tecnologias que otimizem os celulares, tanto em performance, quanto em uso de bateria.

3 A INDÚSTRIA DE CIRCUITOS INTEGRADOS: AVANÇOS TECNOLÓGICOS

Baseado na tecnologia disponível no momento, os computadores foram caracterizados por sua geração, na primeira geração houve advento das válvulas termiônicas, seguida por transistores e, na terceira geração, os circuitos integrados.

O processo de fabricação de um microchip será citado como posterior aos circuitos integrados, porém, o termo "microchip" foi frequentemente utilizado de forma intercambiável com "circuito integrado", mas no contexto da evolução tecnológica, representou uma era de avanços significativos na densidade e miniaturização de componentes.

3.1 A ERA DAS VÁLVULAS TERMIÔNICAS

Patenteado em 1904 pelo seu inventor, o inglês, John Ambrose Fleming (PALLARDY, 2023), o tubo de diodo a vácuo, passou por aprimoramentos, principalmente após a adição da grade de controle, por Lee de Forest, permitindo o controle do fluxo de corrente que percorre o vácuo do tubo (HERMAN, 2001).

Partindo deste princípio, na década de 1940 está válvula, também conhecida como válvula triodo foi utilizada como interruptor para criar os primeiros computadores eletrônicos, dos quais seus predecessores, utilizavam relés eletromecânicos para a função de bloqueio da corrente (HERMAN, 2001).

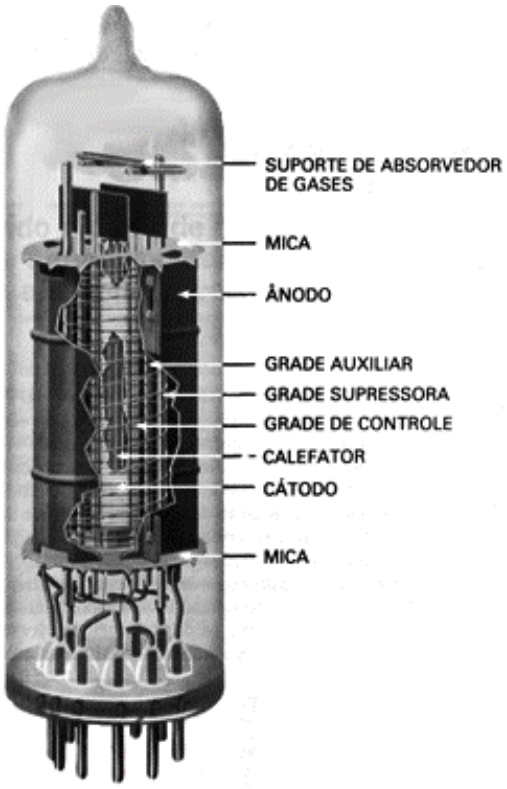
Na Figura 7 é apresentada a válvula termiônica, também conhecida como tubo de triodo a vácuo funciona com base no efeito termoiônico, descoberto por Thomas Edison, que ocorre pela emissão de elétrons em uma superfície metálica, quando aquecida a certas temperaturas (ANDRADE; NETO; LEMAIRE; CRUZ, 2023).

Estes elétrons, agitados pela alta temperatura, conseguem vencer a barreira superficial do metal e viajam através do vácuo dentro da válvula até um ânodo (placa metálica positiva). A grade de controle, fica entre o cátodo e o ânodo, que correspondem aos terminais positivo e o negativo, respectivamente, esta grade, quando carregada negativamente, repele os elétrons que estão viajando através do vácuo, o que desliga o interruptor (ANONYMOUS, 2021).

Este fluxo de elétrons pode ser controlado e usado para amplificar, modificar ou comutar sinais elétricos, sendo fundamental nos rádios e nos primeiros computadores eletrônicos, na época, amplamente usados para fins militares. Contudo, essas válvulas eram

limitadas devido ao seu tamanho, consumo de energia, produção de calor e fragilidade (CARDI, 2002).

Figura 7 - Válvula duplo-triodo.



Fonte: (CARDI, 2002).

As limitações dos componentes dos computadores de tubos a vácuo também limitavam a disseminação dos computadores eletrônicos, principalmente computadores de uso geral.

O primeiro computador de uso geral a ser comercializado foi o UNIVAC I (Figura 8), projetado em 1951, que ocupava cerca de 35,5 m² e pesava cerca de 13 toneladas (Porto Editora). O tamanho físico e o investimento financeiro necessário para possuir um computador eram muito elevados, o que distanciava os computadores da população no geral.

Figura 8 - UNIVAC I.

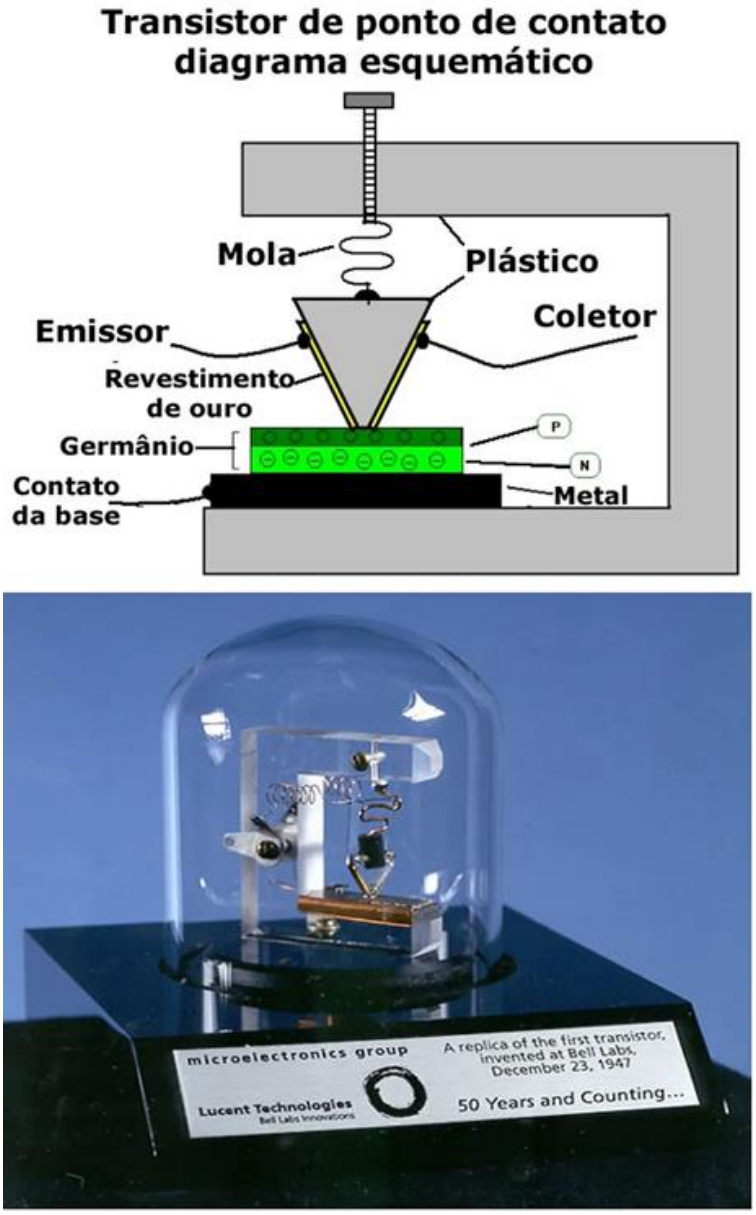
Fonte: (CARDI, 2002).

Com cada inovação em tecnologia os computadores tendiam a diminuir de tamanho e a indústria de manufatura viabilizava a produção em maiores quantidades. A próxima grande inovação tecnológica, vencedora de um prêmio Nobel, viria a ser a criação dos transistores, tratada no capítulo seguinte.

3.2 O SURGIMENTO DOS TRANSISTORES, A SEGUNDA GERAÇÃO

Em 1947, William Shockley, John Bardeen, e Walter Brattain da *Bell Laboratories*, um braço de pesquisa e desenvolvimento de longa data da AT&T (*American Telephone and Telegraph Company*) (ASHBURN, 2023), desenvolveram o transistor (Figura 9), que com 1 centésimo do tamanho de um tubo à vácuo, podia controlar grandes correntes elétricas sem sobreaquecer (HERMAN, 2001).

Figura 9 - Diagrama esquemático do transistor e réplica do primeiro exemplar.



Fonte: (PIROPO, 2012).

A criação do transistor impulsionou a indústria de computadores, o que não só teve em seu principal componente o tamanho drasticamente reduzido, mas também teve a confiabilidade e velocidade aumentadas. No computador o transistor, assim como a válvula, funciona como um interruptor para controlar a execução de operações lógicas (HERMAN, 2001).

Em 1954 o primeiro modelo de computador totalmente de transistores foi o TRADIC (*Transistor Digital Computer*), da *Bell Laboratories*, alguns anos depois, em 1959 foi produzido o $PDP-1$ (*Programmed Data Processor-1*), que viria a ser o hardware a dar vida ao primeiro jogo eletrônico em minicomputadores, *Spacewar*, de Steve Russell (HERMAN, 2001).

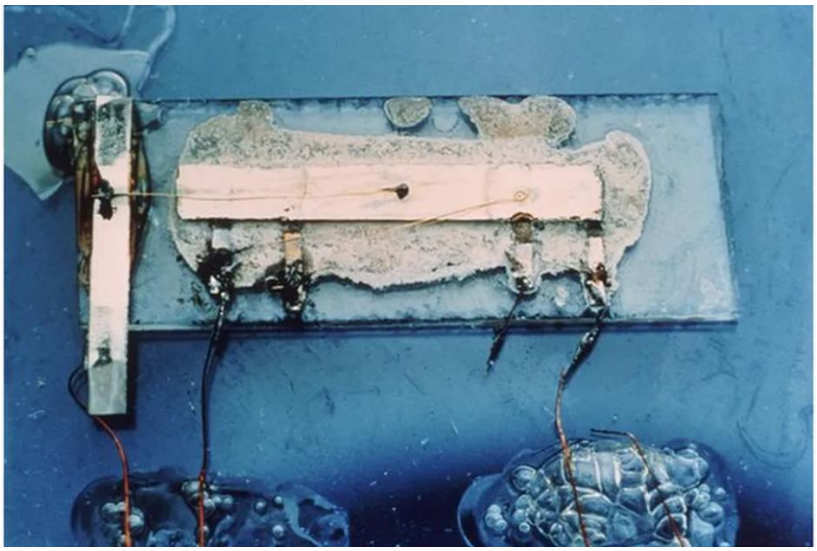
Os transistores permitiram a criação de sistemas com capacidades de computação, porém com custo baixo, como por exemplo o primeiro rádio transistorizado do Japão, o TR-55 (*Transistor Radio 1955*), lançado em 1955 pela *Tokyo Telecommunications Laboratory*, que licenciou a patente da *Bell Laboratories* no ano anterior. Alguns anos depois do lançamento, esta empresa alterou seu nome para Sony (IBUKA, 1998).

3.3 EVOLUÇÃO PARA CIRCUITOS INTEGRADOS

Como pode ser observado na Figura 10, os transistores iniciais necessitavam de muitas soldas, o que aumentava as chances de falhas ocasionadas por erros em sua produção. Com o avanço das pesquisas nestes semicondutores começou um período de miniaturização da eletrônica e produção em larga escala iniciada pelos circuitos integrados, que consistiam em transistores, resistores, capacitores e outros componentes eletrônicos fabricados em um único substrato de material semicondutor, geralmente o silício.

A demonstração dos componentes em um único material semicondutor foi feita por Jack Kilby, enquanto trabalhava na Texas Instruments. Robert Joyce, inovou a fabricação dos circuitos integrados com o método “planar”, com o resultado de seu modo de produzir ilustrado na Figura 11, permitindo a produção em massa destes circuitos (BRAGA, [s.d]).

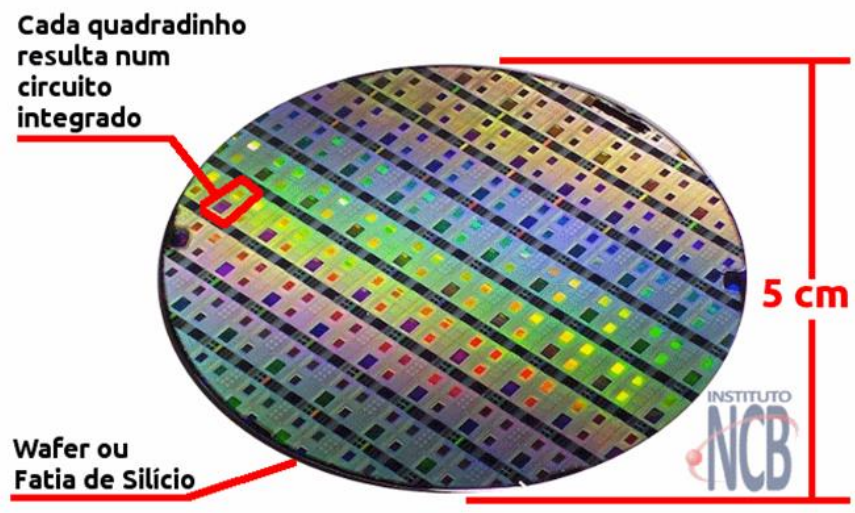
Figura 10 - Protótipo do primeiro circuito integrado, feito por Jack Kilby.



Fonte: (PIROPO, 2012).

Ambos são creditados como coinventores dos circuitos integrados. Como a conexão entre os transistores e outros componentes é feita durante o processo de fabricação do chip, e não por soldagem manual ou mecânica, a quantidade de soldas necessárias é muito reduzida, sendo feita apenas na fixação do circuito integrado no invólucro já preparado para receber o chip (BRAGA, [s.d]).

Figura 11 - Microchips fabricados em um *"Wafer"* de silício semicondutor.



Fonte: (BRAGA, [s.d]).

Com a mudança na criação dos chips, sua fabricação passa a ser produzida em escala maior, partindo de SSI (*Small Scale Integration*), integrando algumas dezenas de transistores

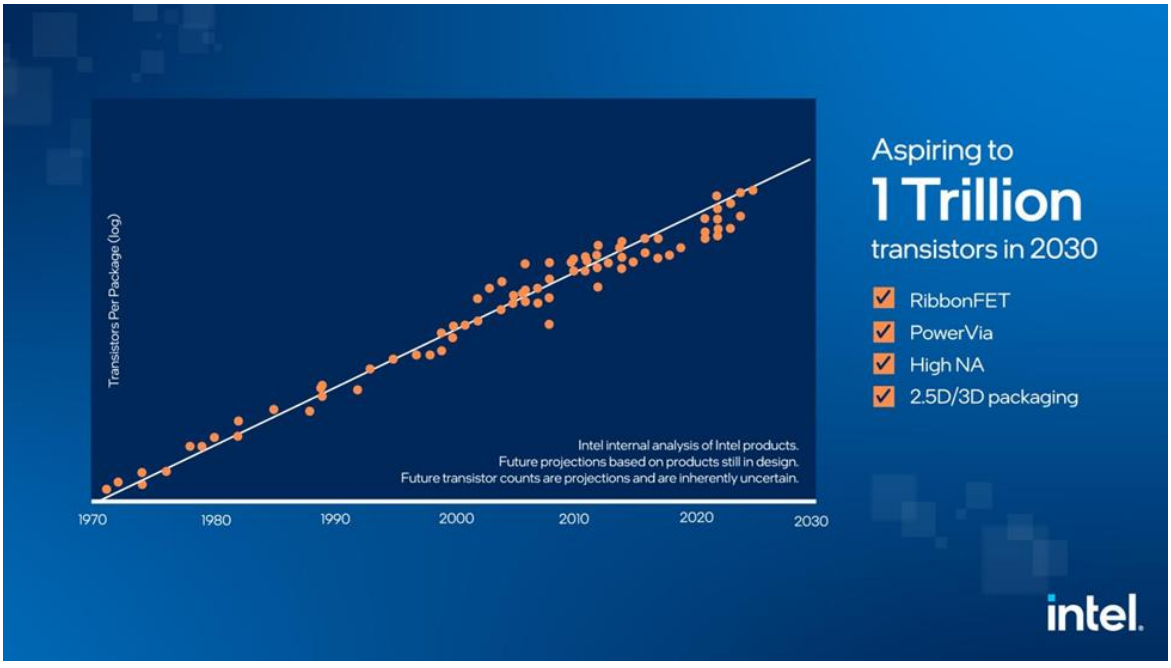
em um único chip, para MSI (*Medium Scale Integration*), com centenas de transistores e LSI (*Large Scale Integration*) com milhares (CARDI, 2002).

3.4 MICROCHIPS

A era do microchip, além da era da informática pessoal é a era da VLSI (*Very Large-Scale Integration*), integrando milhões de transistores em um único chip e, posteriormente, da ULSI (*Ultra Large Scale Integration*), de milhões até bilhões de transistores em um único chip, utilizada nos computadores modernos (CARDI, 2002).

O aumento na escala de produção está ligado à quantidade de transistores em cada pastilha de silício, este aumento segue uma lei empírica denominada lei de Moore, proposta por Gordon Moore em 1965, esta lei estabeleceu que o número de componentes em uma pastilha de silício dobraria a cada 2 anos, aproximadamente (BRAGA, [s.d]). Conforme Figura 12, demonstração da Intel sobre a lei de Moore, segundo a empresa, essa lei segue aplicável atualmente, com projeções de mantê-la até 2030.

Figura 12 – Projeção da Intel, mostrando passado, presente e futuro em relação à lei de Moore.



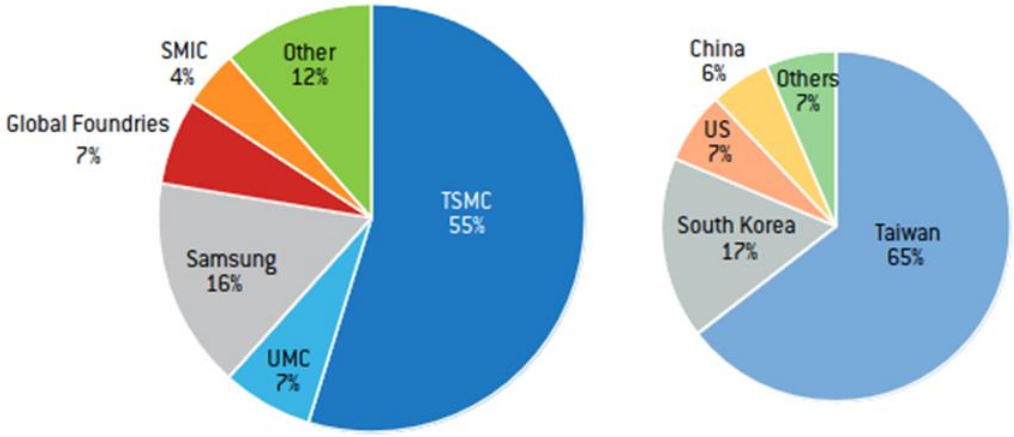
Fonte: (KELLEHER, 2022).

A manutenção da Lei de Moore depende do avanço tecnológico na produção de microchips, no próximo subtópico serão estudados alguns fatores que mantém essa projeção real até os dias atuais.

3.5 A MANUFATURA DOS CIRCUITOS INTEGRADOS

A fabricação de microchips atualmente é um processo dominado por poucas empresas, cinco empresas, situadas em quatro países, detêm quase 90% do mercado de fabricação de semicondutores, como ilustrado na Figura 13.

Figura 13 - Fabricantes de semicondutores no mundo.



Fonte: (POITIERS, 2021).

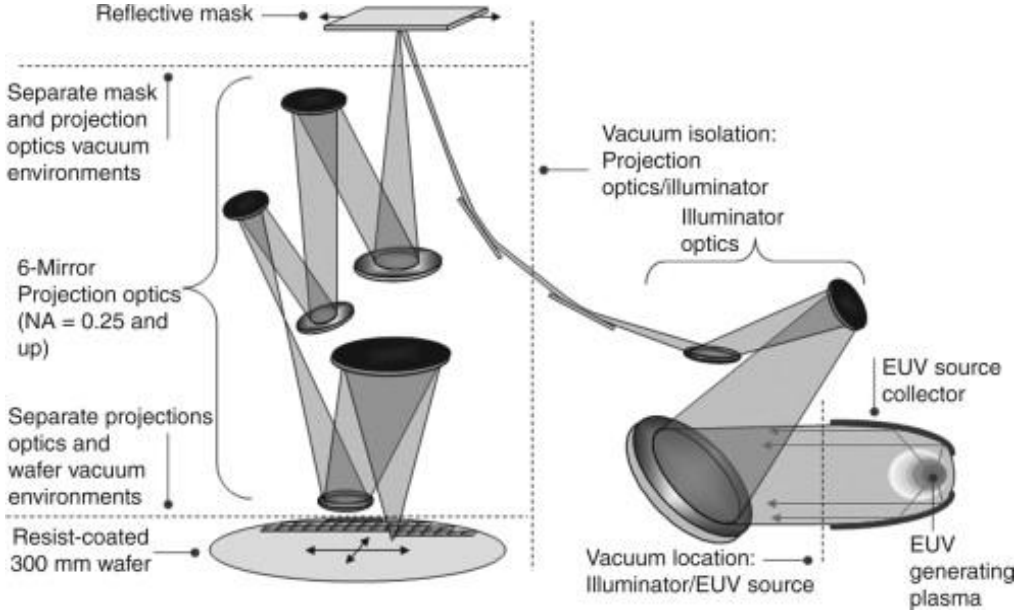
Destaca-se TSMC (*Taiwan Semiconductor Manufacturing Company*) e Samsung, que são as únicas capazes de fabricar chips com nó de 10 nm e abaixo. A TSMC lidera em termos de capacidade de fabricação (POITIERS, 2021).

3.6 OS AVANÇOS NA LITOGRAFIA DOS CIRCUITOS INTEGRADOS

Intel, *Samsung* e TSMC, são os principais clientes da holandesa ASML, que domina o mercado de construção das máquinas que fazem os microchips com tecnologia de litografia EUV (*Extreme Ultraviolet*). A tecnologia EUV, possui comprimento de onda de 13,5 nm, 15 vezes menor que sistemas de 193 nm, predominantes no mercado à época (FELDMAN, 2014). A litografia EUV utiliza materiais altamente reflexivos e transmissão de fótons no vácuo, como ilustrado na Figura 14, pois 13,5 nm está na banda do raio x mole, não sendo possível usar óticas refratárias em razão da absorção dos fótons pelos materiais e o ar (FELDMAN, 2014). Com avanços na tecnologia EUV atualmente é possível fazer um transistor funcional com nó de 2 nm, mesmo que ainda não seja possível fabricar um dispositivo completo com nó deste tamanho. Um dispositivo assim, em relação a dispositivos

com 7 nm, como o smartphone iPhone 11, poderia quadruplicar o tempo com bateria, o que pode tornar a recarga da bateria do celular necessária apenas uma vez a cada quatro dias, com uso médio (FROUGIER, 2021).

Figura 14 - Subsistemas chave da litografia EUV

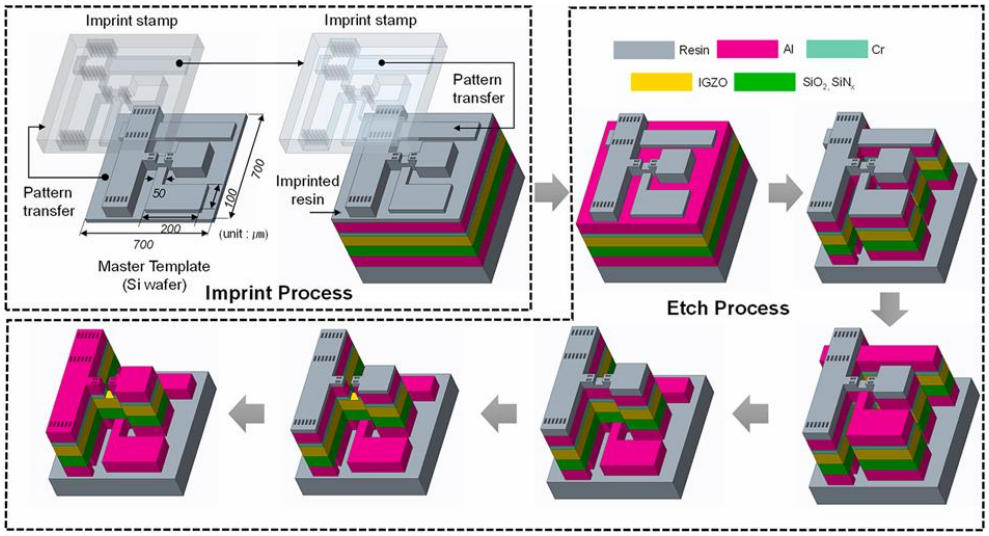


Fonte: (FELDMAN, 2014).

O processo de litografia, por si só, é impressionantemente inovador, base na produção de microchips é usado para transferir padrões precisos para a superfície de um "*wafer"* de silício, e como visto, busca diminuir, cada vez mais, o espaço entre conexões, os nós. Na Figura 15 é demonstrada uma técnica chamada SAIL (*Self-Aligned Imprint Lithography*), que usa um método *"top-down"* para formar a estrutura desejada do dispositivo, com a deposição de todas as camadas de filme fino no substrato, e após, de cima para baixo remove as partes desnecessárias por meio de uma máscara de gravação (em cinza, na Figura 15), evitando o desalinhamento de padrões que pode ocorrer nos processos convencionais, que tradicionalmente utilizam um método *"bottom-up"* (JEON; CHO; CHO, 2021).

O tema da fabricação de microchips é bastante amplo e envolve, em sala limpa, processos de dopagem, adsorção, corrosão úmida, corrosão seca, oxidação, fotolitografia, entre vários outros, além de seu peso geopolítico, no centro de disputas entre potências mundiais.

Figura 15 – Sequência esquemática do processo de litografia "*top-down*"



Fonte: (JEON; CHO; CHO, 2021).

Muitas fontes acadêmicas fornecem informações aprofundadas sobre a história e evolução dos microprocessadores, é um tema complexo, inovador e amplo, motivo de diversos estudos (ALVES, 1999).

4 O DESENVOLVIMENTO DA LINGUAGEM DE MONTAGEM (ASSEMBLY) E SUA APLICAÇÃO EM JOGOS DIGITAIS

Como visto, cada microprocessador pode possuir bilhões de transistores, que por meio de entradas e saídas binárias, apresentam conjuntos de instruções a serem executadas durante um ciclo de instruções. Essas entradas binárias executam ações pré-definidas pelo seu *hardware* que, mesmo que mais complexas, podem ser comparadas à tabela-verdade de uma porta lógica. O microprocessador reconhece apenas padrões de entradas binários, outros padrões como decimal ou hexadecimal não são reconhecidos, apesar de serem utilizados por programadores por facilitar o entendimento dos códigos escritos (WELSH; KNAGGS, 2003).

As instruções do microprocessador são executadas sequencialmente de forma crescente nos endereços de memória, exceto quando explicitado para não o fazer, ou seja, o microprocessador obtém a próxima instrução do próximo endereço de memória, ao menos que a próxima instrução o faça executar a instrução de outro endereço, que não o da sequência (WELSH; KNAGGS, 2003).

4.1 MICROPROCESSADORES E A EFICIÊNCIA DA BASE HEXADECIMAL

No microprocessador, as instruções são interpretadas em base binária (base 2), uma característica que está intrinsecamente ligada à arquitetura do hardware. Esta abordagem binária regula a passagem de corrente elétrica, correspondendo aos comandos definidos pelo programador no código do programa. A representação direta em base 2 pode ser desafiadora para a compreensão humana, aumentando a suscetibilidade a erros durante a escrita e a depuração do código, além de resultar em sequências extensas de dados, portanto, escrever os códigos em base 8 ou base 16, octal e hexadecimal, respectivamente, torna o código mais curto. A base hexadecimal é o padrão da indústria de microprocessadores (WELSH; KNAGGS, 2003).

Em (1), através de código em linguagem de montagem x86, foi feita a soma entre dois números em base hexadecimal e o seu resultado alocado no registrador “ah”. Nota-se na Figura 16, em verde, que o código completo desta soma é o mesmo código hexadecimal em (2).

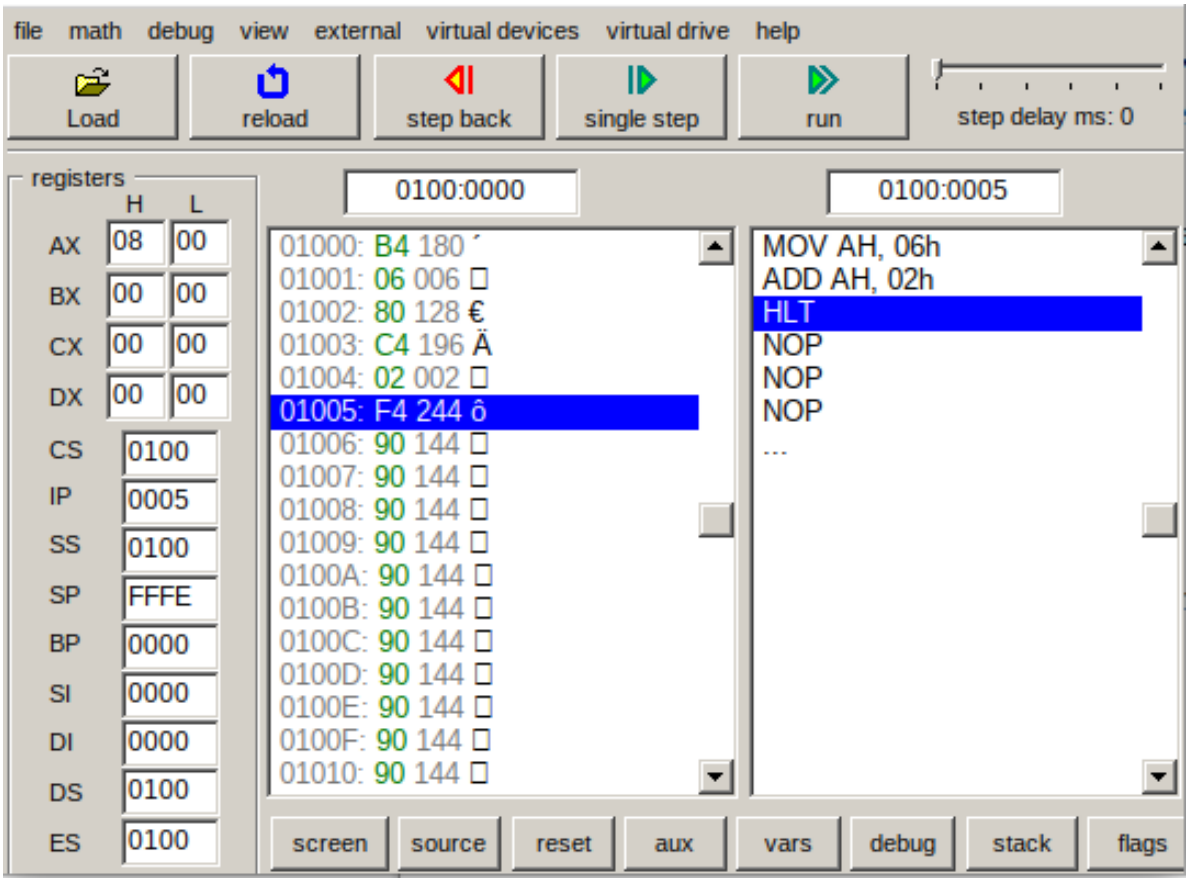
```
MOV AH 06h ADD AH 02h HLT                                    (1)
```

$B4\ 06\ 80\ C4\ 02\ F4.$ (2)

O mesmo código escrito em base binária (3), é muito maior.

$$\underset{\overset{\smile}{MOV}ah}{10110100}\ \underset{\overset{\smile}{06}h}{00000110}\ \underset{\overset{\smile}{ADD}}{10000000}\ \underset{\overset{\smile}{a}h}{11000100}\ \underset{\overset{\smile}{02}h}{00000010}\ \underset{\overset{\smile}{hlt}}{11110100} \quad (3)$$

Figura 16 - Conjunto de instruções para realizar uma soma em um processador 8086. Utilizou-se o emulador EMU8086.



Fonte: Elaboração própria.

Na Figura 16, observa-se inicialmente que o processador é instruído a mover o valor hexadecimal “06” para o registrador “ah”. Em seguida, ele executa a operação de soma do valor hexadecimal “02” ao valor anteriormente armazenado no registrador “ah”. O resultado dessa soma, “08” em hexadecimal, é armazenado no mesmo registrador “ah”. A Figura 16 também exibe, em seu quadro mais à direita, o mesmo processo de soma, mas desta vez utiliza-se abreviações como “MOV”, “ADD” e “HLT”. Para a execução correta dessas

operações são utilizados mnemônicos[2], é necessário conhecer não apenas as instruções e os valores a serem manipulados no programa, como “MOV” e “02h”, mas também os registradores específicos, como “ah”.

4.2 AS FUNÇÕES DA LINGUAGEM DE MONTAGEM

A linguagem de montagem utiliza os mnemônicos, sendo armazenada como texto, assim como linguagens de alto nível, o que a faz ponte fundamental entre a programação legível por humanos e a linguagem de máquina interpretada por microprocessadores (WELSH; KNAGGS, 2003).

Utilizar nomes e endereços simbólicos facilita muito o processo de criação de código, pois tanto para instruções, quanto para endereços de memória, números binários e hexadecimais carregam maiores chances de cometer erros (WELSH; KNAGGS, 2003).

Mesmo que facilite o processo de escrita e entendimento do código, um código escrito em linguagem de montagem precisa ser transformado em binário para execução do hardware, essa tradução era conhecida como montagem manual *(“hand assembly”)* (WELSH; KNAGGS, 2003).

A montagem manual envolve a conversão de cada instrução da linguagem de montagem para o seu equivalente em hexadecimal ou binário, o que manteve certo grau de erros e ineficiência, pois, apesar de diminuir falhas por elevar a compreensão da linguagem, este processo mantinha o problema de erro manual, agora na transcrição do código para binário (WELSH; KNAGGS, 2003).

Entre 1951 e 1952 Nathaniel Rochester desenvolveu o que foi considerado o primeiro montador, para o IBM 701 (ROCHESTER, 1983). O montador foi um programa que automatizou o processo de transcrição de linguagem de montagem para código de máquina, sem cometer erros, o que proporcionou eficiência e confiabilidade. Em nível baixo[3], para cada

---

[2]Mnemônicos, segundo o dicionário online Michaelis, “Diz-se de técnica ou exercício que ajuda a desenvolver a memória e facilita a memorização,” e “Fácil de se reter na memória.”. No contexto de programação em *assembly*, os fabricantes de microchips utilizam seu próprio conjunto de mnemônicos para simplificar o nome das instruções do componente em linguagem de montagem, como por exemplo a instrução para interromper o programa em execução, o “HLT”, abreviação de “*halt*”, palavra de língua inglesa que significa parar ou interromper.

[3]Na programação, as linguagens e seus tradutores são categorizados com base na proximidade da linguagem de máquina. Os montadores são considerados de baixo nível. Em contraste, as linguagens de alto nível, como *Java* ou *C*, estão mais distantes da linguagem de máquina, consideram complexidades e são traduzidas para a

hardware, há uma linguagem de montagem e pode haver um montador. Para linguagens de nível alto, a tradução é feita por um compilador, que traduz a linguagem de nível alto para conjuntos de instruções em linguagem de montagem. Por possuir maior abstração na escrita, linguagens de nível alto permitiram a descrição de tarefas de forma mais orientada ao problema, do que orientada ao hardware, o que agilizou a solução de problemas e trouxe produtividade ao mercado de programação (WELSH; KNAGGS, 2003; TANENBAUM, 2013).

Com o advento das linguagens de nível alto, a linguagem de montagem perdeu popularidade entre os programadores, principalmente pelos avanços de hardware, o que permitiu a criação de *softwares* menos otimizados em menor tempo de produção. Ainda assim existem vantagens para o uso da linguagem de montagem, como a criação de compiladores e depuradores, acesso a hardwares e drivers de sistemas, diminuir o tamanho do código e programação de sistemas embarcados (STALLINGS, 2017).

Na educação, em comparação com linguagens de nível alto, a utilização da linguagem de montagem traz diversos benefícios, com ela é possível esclarecer o acesso a dispositivos externos, o modo como o sistema operacional interage com o programa, a representação dos dados na memória e a execução de instruções no hardware. Outro ponto é que sem entender as bases históricas e teóricas da linguagem e do hardware, parte do entendimento da criação e da evolução é perdido, o que proporciona um ensino incompleto sobre o tema (STALLINGS, 2017).

Como mencionado anteriormente, cada microprocessador possui seus códigos de instruções, baseados na arquitetura de seu hardware, a arquitetura que foi usada neste trabalho é a x86, da Intel, família de microprocessadores muito famosa e predominante, principalmente por apresentar retro compatibilidade em seus novos lançamentos de microprocessadores. Essa arquitetura, assim como diversas outras são fortemente influenciadas pela arquitetura proposta por John Von Neumann, essa ideia de arquitetura foi publicada em 1945, na criação do computador EDVAC (*Electronic Discrete Variable Computer*), baseada no conceito de programa armazenado. Em 1946, Von Neumann e equipe começaram a desenvolver um computador de programa armazenado, em Princeton, o computador foi denominado IAS (*Institute for Advanced Studies)*, finalizado em 1952, foi o protótipo de uma arquitetura da qual a maioria dos computadores de propósito geral usa (STALLINGS, 2017).

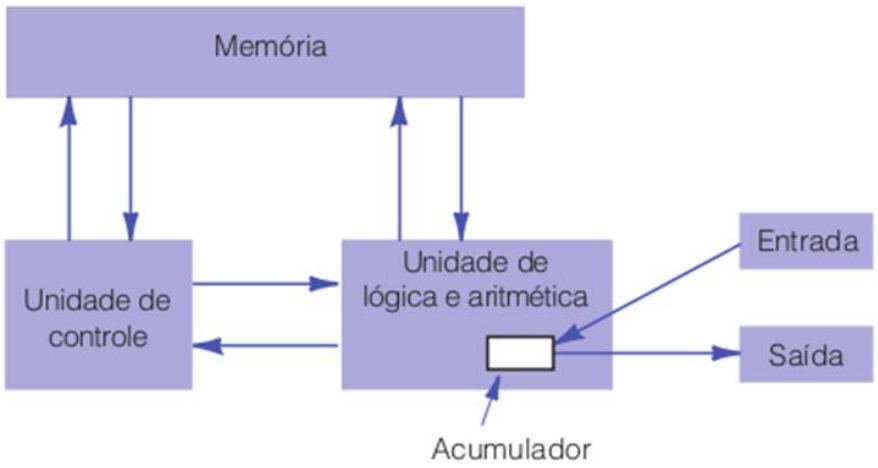
---

linguagem de máquina por compiladores. A nomenclatura ilustra a decisão entre a facilidade de uso para o programador e o controle direto do hardware (TANENBAUM, 2013).

4.3 A ARQUITETURA DE VON NEUMANN

A arquitetura desenvolvida por Von Neumann (Figura 17) revolucionou a estrutura dos computadores e, por extensão, das linguagens de programação. Ao construir separadamente a CPU (*Central Processing Unit*) e a memória principal, Von Neumann simplificou a arquitetura do computador e aumentou a eficiência de processamento. Nesta construção de computador, com possibilidade de transferência de informações por entradas e saídas, permite-se sequências longas de dados e instruções, por serem armazenados na memória principal. A sequenciação adequada destes dados e instruções é feita pela UC (Unidade de Controle) do dispositivo e as operações elementares (adição, subtração, multiplicação e divisão), que ocorrem com maior frequência, tem uma ULA (Unidade Lógica Aritmética), que é especializada nessas operações. Vale ressaltar que a ULA e UC, em computadores modernos, são combinadas no mesmo chip e identificadas com CPU (STALLINGS, 2017).

Figura 17 - Estrutura da máquina de Von Neumann.



Fonte: (TANENBAUM, 2013).

Esta estrutura ou parte dela foi replicada na imensa maioria dos computadores atuais e, como mencionado anteriormente, na arquitetura x86, que foi utilizada neste trabalho. A família de arquiteturas x86 foi projetada para ter compatibilidade binária com modelos antigos, o que permitia manter os *softwares* de diferentes gerações de modelos, esse ponto foi chave para sua disseminação.

4.4 A LINGUAGEM DE MONTAGEM E SUA APLICAÇÃO NOS JOGOS DIGITAIS

Parte fundamental da computação moderna, a linguagem de montagem desempenha um papel crucial no desenvolvimento de jogos digitais, especialmente ao se considerar a programação de hardware com recursos limitados, como os sistemas de videogame.

4.4.1 VÍDEO JOGOS E COMPUTADORES

Os primeiros jogos digitais, como os icônicos *"Pong"* e *"Space Invaders"* do Atari 2600 foram moldados em linguagem de montagem, para o microprocessador. Estando um degrau acima do código de máquina, a linguagem de montagem é profundamente ligada ao hardware, permitindo que programadores explorem o potencial completo dos processadores da época, utilizando o máximo possível do hardware, mesmo que tenham recursos limitados, como por exemplo condensar um jogo em quantidades de memória extremamente baixas. Um cartucho de Atari possuía de 1 a 4 KB de memória ROM (*Read-Only Memory*) contendo o jogo (BRANDÃO), já *"Starfield",* lançado em 2023, ocupa 125 GB de armazenamento (CARBONE, 2023).

Originalmente projetados para entretenimento básico em aparelhos de TV, os sistemas de jogos digitais expandiram suas capacidades, atingindo um nível de desempenho que desafia e, em alguns casos, supera os computadores pessoais tradicionais. Esta evolução não apenas reflete o avanço tecnológico significativo, mas também ressalta o papel crescente dos jogos na cultura digital contemporânea (TANENBAUM, 2013).

Consoles como o PlayStation 3 da Sony e o Xbox 360 da Microsoft, foram exemplos dessa evolução. Equipados com processadores de múltiplos núcleos e GPUs (*Graphics Processing Units*) avançadas, estes dispositivos foram otimizados para oferecer uma experiência de jogo em 3D e multimídia de alta qualidade. Diferentemente dos PCs (*Personal Computers*), os consoles são projetados como sistemas integrados e focados, com menos possibilidades de expansão, mas com otimizações específicas para jogos e entretenimento (TANENBAUM, 2013).

A abordagem de sistema fechado nos consoles de jogos teve suas limitações, como a menor capacidade de expansão e personalização, contudo, essa estratégia possibilitou a produção e venda a custos mais acessíveis, o que contribuiu para a popularidade massiva desses dispositivos (TANENBAUM, 2013).

No contexto dos dispositivos móveis, o desafio foi balancear o desempenho com o consumo eficiente de energia. Enquanto buscou-se fornecer capacidades gráficas e de processamento comparáveis aos dos PCs, *tablets* e *smartphones* também precisavam otimizar o uso de energia para maximizar a vida útil da bateria. Essa necessidade representa um desafio complexo de engenharia, equilibrando desempenho e eficiência energética. Este é um motivo, dentre outros, de atualmente a arquitetura mais comum em *smartphones* ser a ARM[4] (*Advanced RISC Machine*), com conjunto de instruções mais simples, em RISC[5] (*Reduced Instruction Set Computer*) (TANENBAUM, 2013).

4.5 ALÉM DA EFICIÊNCIA: LEGIBILIDADE E PORTABILIDADE

Eficiência e precisão são apenas partes de um conjunto maior de considerações no desenvolvimento de software. Fatores como legibilidade, escalabilidade e portabilidade também são cruciais. Os compiladores atuais são capazes de traduzir código-fonte de alto nível em uma linguagem de máquina altamente otimizada em diversas arquiteturas de processadores.

A capacidade avançada de otimização atual faz com que, na maioria dos casos, a necessidade de escrever diretamente em linguagem de montagem seja eliminada, já que os compiladores atuais conseguem gerar um código eficiente que, em muitos casos, se equipara ao desempenho que seria obtido se o mesmo código fosse escrito diretamente em linguagem de montagem (SEVERANCE).

Com acesso direto ao *hardware*, principalmente as entradas e saídas, é possível manipular diversos componentes pelo PC, como por exemplo, manipular os pixels de um monitor através de um cabo VGA (*Video Graphics Array*), o próximo capítulo busca demonstrar o uso desta tecnologia, chave para o conceito de TV interativa que culminou nos videojogos. Para acessar e interagir com esta parte do CPU, diversos conceitos foram aplicados, todos foram descritos no próximo capítulo.

---

[4]A tecnologia ARM (*Advanced RISC Machine*) é uma tecnologia de instrução mais simples e limitada, porém isso permite que processadores RISC sejam mais eficientes em uso de energia (ROUSE; OGDEN, 2004).

[5]A arquitetura de processador RISC (*Reduced Instruction Set Computing*) começou a aparecer na década de 1980 contendo menos instruções e modos de endereçamento da memória em relação à arquitetura CISC (*complex instruction set computer)*, que já era presente e chegava a demandar diversos ciclos de *clock* para executar instruções (ROUSE; OGDEN, 2004).

5 APLICAÇÃO DOS ELEMENTOS DO JOGO EM AMBIENTE DE SIMULAÇÃO

Os elementos de jogo deste trabalho foram inspirados no livro de Oscar T. Gutierrez, *Programming Boot Sector Games* (2019), no qual o autor cria jogos de setor de inicialização e explica o passo a passo para tal. Também houve influência do código escrito por Kyle Vassau, que, também inspirado pelo livro de Oscar Gutierrez, criou versões para seus próprios jogos (VASSAU, 2023).

A ideia deste trabalho foi desenhar, em uma tela, uma nave que respondesse a comandos do jogador para se movimentar. O ambiente do jogo foi o mesmo da época do lançamento do Atari 2600, para isso utilizou-se a linguagem de montagem dos microprocessadores 8086/88, lançados em épocas próximas. O jogo interativo foi implementado na linguagem de montagem x86 e executado em um ambiente de setor de inicialização. O código foi organizado em várias rotinas e subrotinas, cada uma desempenhou uma função específica essencial para executar o programa, o código-fonte do jogo encontra-se no Apêndice A. A seguir, apresenta-se uma visão geral dos pontos tidos como principais para este trabalho.

5.1 SELEÇÃO DO AMBIENTE DE DESENVOLVIMENTO E ELEMENTOS DO JOGO

A fim de simular um ambiente com arquitetura x86 de 16 bits, o 8086 ou 8088 lançados em 1978 e 1979, respectivamente, foram utilizados 2 *softwares* em conjunto, o Vscode[6] (*Visual Studio Code*) e QEMU[7] (*Quick Emulator*).

O Vscode foi utilizado juntamente com as extensões, “nasm-compiler-linux, v0.0.5” que permitiu a compilação e execução do código com linguagem NASM[8] (*Netwide Assembler*), o “makefile v0.7.0” que permitiu uso do utilitário *make*[9] para criar um padrão de execução para

[6]O *Visual Studio Code* (Vscode) é um software de edição de código-fonte disponível para Linux, MacOS e Windows. Por possuir um ecossistema de extensões, foi o editor de código escolhido para o trabalho (Visual Studio Code, 2023).

[7]QEMU é um emulador de máquina e virtualizador genérico e de código aberto. Foi utilizada uma extensão no ambiente Vscode para executar o *makefile* (BELLARD, 2005).

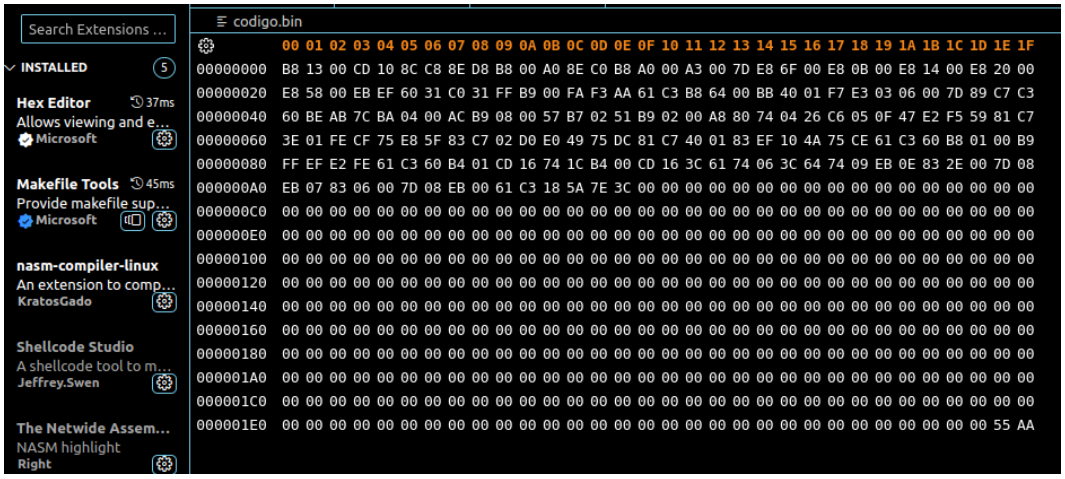
[8]O *Netwide Assembler* é um montador para a linguagem de montagem da arquitetura x86 (the NASM Authors, 2021).

[9]O *Make* opera lendo um arquivo chamado *Makefile*, que especifica como os arquivos executáveis e outros arquivos derivados são gerados a partir dos arquivos fonte. No trabalho foi escrito um *Makefile*, o que permitiu ao *Make* gerenciar eficientemente a compilação e instalação (GNU Make, 2020).

o código em linguagem de montagem x86 e a extensão “Hex Editor v1.9.12” que permitiu a visualização do código em linguagem hexadecimal, exemplificado na Figura 18.

Com ambos os *softwares* foi possível executar o código e obter resposta visual na execução, via modo VGA no QEMU, entre outras funcionalidades explicitadas durante o trabalho.

Figura 18 - Extensões utilizadas e código-fonte do jogo visualizado pela extensão Hex Editor.



Fonte: Elaboração própria.

O código utilizado no *makefile* foi obtido via código fonte de Kyle Vassau, também baseado em setor de inicialização e inspirado no trabalho de Gutierrez. Em seu código, o autor define no arquivo codigo.asm a linguagem nasm, em formato (-f) binário, com saída (-o) de mesmo nome, porém agora em formato binário (codigo.bin). Após, o autor executa a emulação, via QEMU, em sistema i386, arquitetura x86 de 32 bits (VASSAU, 2023).

all:

$nasm\ -f\ bin\ codigo.asm\ -o\ codigo.bin$

$executar$:

$qemu-system-i386\ -drive\ format=raw, file=codigo.bin$

5.2 DESENVOLVIMENTO DA LÓGICA DO JOGO

O primeiro passo foi definir os endereços de memória para serem usados no sistema emulado. O jogo foi inspirado em jogos de setor de inicialização, portanto houve 512 bytes disponíveis para uso, que correspondem aos primeiro 512 bytes usados para inicialização do

sistema, através da BIOS (*Basic Input/Output System*), que é um chip ROM na placa mãe (GUTIÉRREZ, 2019).

5.2.1 O SETOR DE INICIALIZAÇÃO

Ao iniciar o computador, o BIOS tem código suficiente para ler o setor de inicialização na memória RAM, mas o setor de inicialização não tem espaço suficiente para conter todo o sistema operacional, portanto sua função é carregar o sistema operacional na RAM, como Windows ou Linux (GUTIÉRREZ, 2019).

O trabalho explica o uso de cada endereço de memória e seu porquê, pois, ao levar em conta que se busca reproduzir jogos também de consoles, cada console pode ter seu BIOS e modo de inicialização, o que leva a carregar diferentes áreas da memória e instruções para tal (GUTIÉRREZ, 2019).

Nos PCs IBM e compatíveis, quando reiniciado, o processador é direcionado para o endereço FFFF:0000 (CS:IP), ou seja, no *code segment* (CS), o apontador de instruções (IP) aponta para o primeiro endereço de memória, executando as primeiras instruções do BIOS. O BIOS, então, verifica os componentes básicos do sistema, como memórias, contadores, armazenamento e portas (GUTIÉRREZ, 2019).

Após isto, o BIOS tenta inicializar o sistema através de um dispositivo de armazenamento, à época um disquete. O BIOS lê o conteúdo do primeiro setor (512 bytes de memória) do disquete, no endereço 07C00h e checa se os dois últimos bytes do setor contêm a assinatura de inicialização (55h e AAh). Caso haja a assinatura, o BIOS carrega CS com 0 e IP com 7C00h, portanto 0000:7C00 (CS:IP) (GUTIÉRREZ, 2019).

O ambiente de trabalho simula a execução do código em uma memória virtual, portanto algumas adequações foram feitas no programa. Ao prever que após a inicialização o próximo endereço de memória seria 7C00h, o código do jogo foi escrito a partir deste endereço, e não pôde ultrapassar o endereço 7DFDh, pois os dois últimos endereços correspondem à assinatura de inicialização, como citado anteriormente (GUTIÉRREZ, 2019).

A fim de que o código fosse executado de forma correta, o início e o fim do código deveriam conter as seguintes instruções (GUTIÉRREZ, 2019).

$ORG\ 7C00h$

…

$times\ 510\ -\ (\$ - \$\$)\ db\ 0$

$db\ 55h$

$db\ Aah$

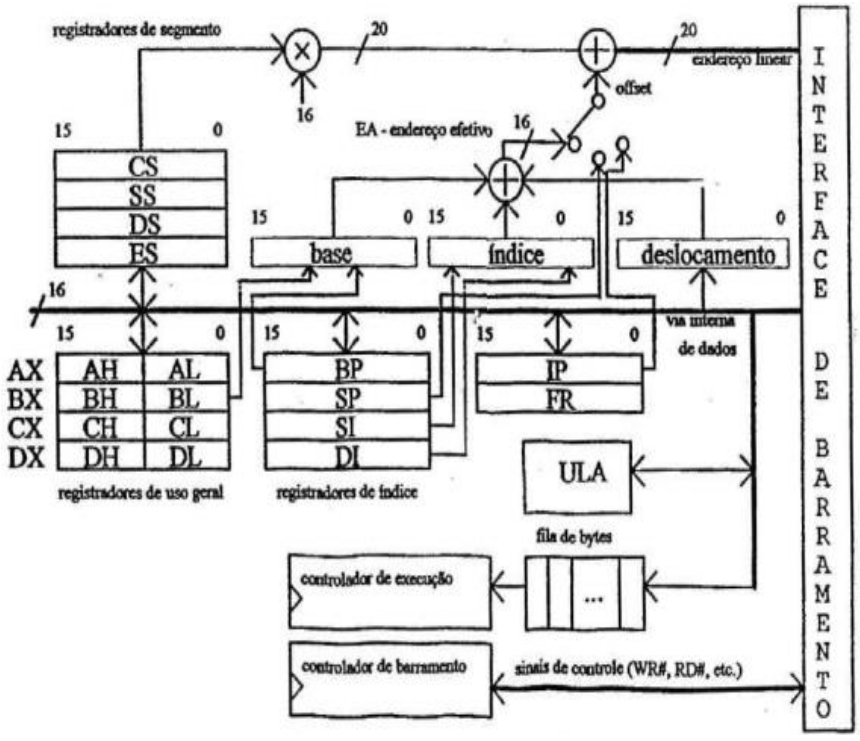
A diretiva “ORG”, define que o código seja escrito a partir do endereço indicado, a instrução "times 510 - ($-$$) db 0" adiciona bytes de valor zero ao programa até que seu tamanho total (excluindo os dois bytes finais) fosse de 510 bytes, isso assegura que há espaço suficiente para que o BIOS possa entender que há 512 bytes e que os últimos 2 bytes são a assinatura de setor de inicialização (BLUNDELL, 2010).

5.2.2 OS REGISTRADORES

Definidos os códigos para o setor de inicialização, foi feito o desenvolvimento do código do programa. Este capítulo explica brevemente a função dos registradores, parte muito importante do código.

Os microprocessadores 8086/88 possuíam 14 registradores de 16 bits, sendo 4 registradores de uso geral (AX, BX, CX e DX), 4 registradores apontadores (BP, SP, SI e DI), 1 ponteiro de instrução (IP – *Instruction Pointer*), 1 registrador de estado (PSW– *program status word*) e 4 registradores de segmento (CS, SS, DS e ES), Figura 19 (HYDE, 2010).

Figura 19 - Arquitetura simplificada do 8086/88



Fonte: (HYDE, 2010)

Os registradores apontadores SP e BP são utilizados para acessar dados no segmento de pilha, os registradores SI e DI facilitam a movimentação de dados através do SI (endereço fonte) para o DI (endereço destino). O registrador de ponteiro de instrução sempre aponta para a próxima instrução a ser executada. O registrador de estado PSW possui 16 bits, em que sete não são utilizados, e os demais são tratados individualmente e indicam algum estado (HYDE, 2010).

5.2.3 A MEMÓRIA E O MODO DE VÍDEO

Como dito anteriormente, há um lugar específico na memória para carregar os dados de inicialização, o mesmo acontece com outros segmentos de endereço de memória (Figura 20), por exemplo o segmento de vídeo, que permite acesso a diversos modos de vídeo, como CGA, EGA e VGA.

Figura 20 - Ilustração da segmentação da memória

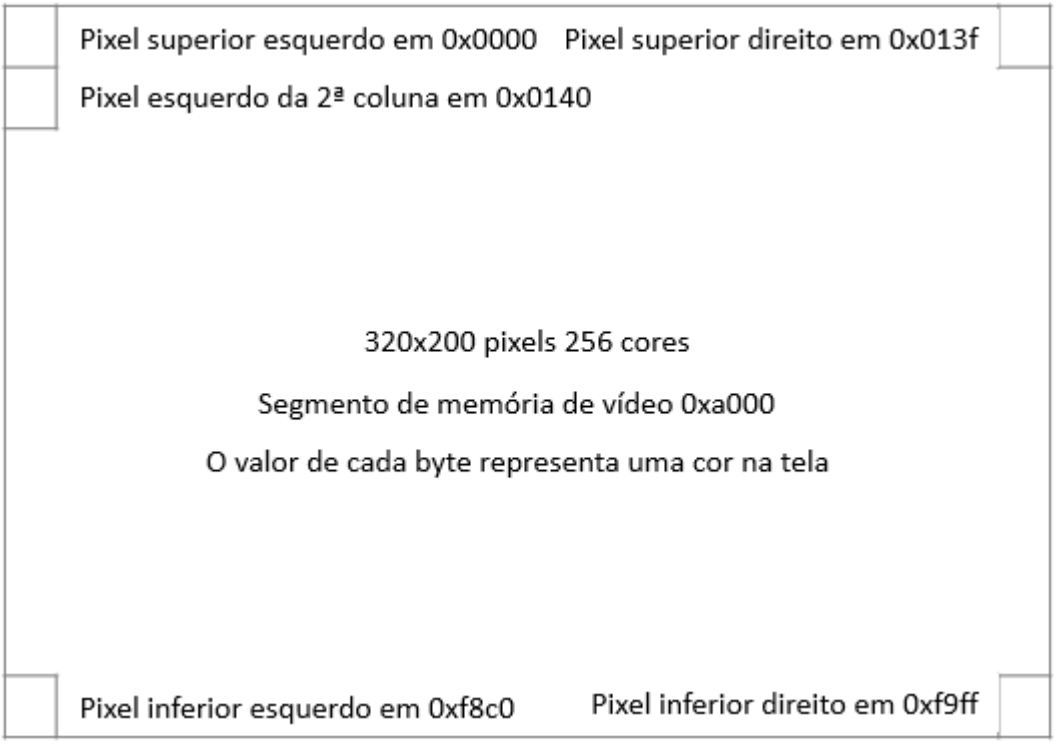
Fonte: Adaptado de (GUTIÉRREZ, 2019).

O modo VGA com resolução de 320x200 pixels e 256 cores foi utilizado neste trabalho, o ponto principal foi o fato deste modo de vídeo possuir tela gráfica planar, o que facilita ao manipular os pixels, pois cada pixel é representado por um único byte na memória. Com a resolução de 320x200 pixels, há 64000 pixels, ou seja, 64000 bytes para manipular a tela. O

endereço para memória de vídeo VGA é o A000:0000, para o primeiro pixel, e A000:F9FF, para o último pixel. Nota-se que o segmento de memória VGA termina em A000:FFFF, portanto existe uma parte da memória de vídeo que não é visível graficamente (1536 bytes), mas ainda assim pode ser utilizada para armazenar dados (GUTIÉRREZ, 2019).

Na tela gráfica planar o primeiro pixel no topo, à esquerda, é equivalente ao primeiro byte da memória A000:0000. Ele é seguido pelo próximo pixel à direita (primeira linha, segunda coluna), representado pelo segundo byte A000:0001 e assim sucessivamente até completar a linha no pixel 320 ou A000:013f, recomeçando na segunda linha, primeira coluna, no endereço A000:0140, dessa forma até o último pixel, localizado no endereço A000:F9FF (Figura 21) (GUTIÉRREZ, 2019).

Figura 21 - Endereço de cada pixel em uma tela VGA 320x200 pixels, com 256 cores.



Fonte: Adaptado de (GUTIÉRREZ, 2019).

A forma de representação em duas dimensões facilitou o entendimento e manipulação dos pixels ao simplificar a visualização do jogo e consequentemente dos pixels na tela.

5.2.4 *INSTRUÇÃO "EQU"*

Anteriormente, foi citada a importância da linguagem de montagem para facilitar e dinamizar o trabalho do programador, o que a manteve em constante atualização. Um importante aprimoramento foi a criação da instrução "EQU", que atribuiu valores absolutos ou relocáveis a rótulos, retornando o valor definido toda vez que for utilizado este rótulo (HYDE, 2010).

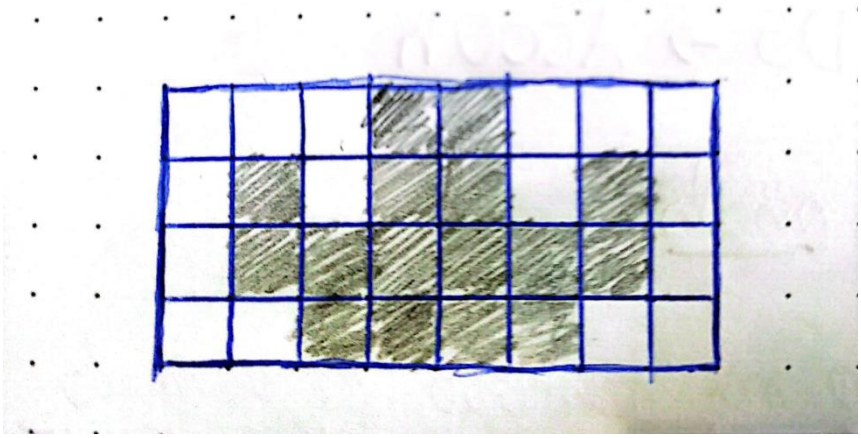
Esta instrução permitiu que fossem criados nomes para constantes e variáveis, dessa forma ao tentar acessar um endereço de memória que continha um bitmap necessário ao jogo, por exemplo, digitou-se o nome já definido anteriormente, e o montador fez a substituição para executar o programa.

5.2.5 *BITMAPS*

Neste trabalho também foram usados *bitmaps*. Os *bitmaps* foram fundamentais na representação gráfica em computação, principalmente em programação de jogos em setor de inicialização. A seguir, está a representação da nave do jogo em *bitmaps* e na figura 22 o desenho que representou a ideia da nave em jogo.

```
00011000
01011010
01111110
00111100
```

Figura 22 – Desenho, feito à mão em folha pontilhada, do *bitmap* desejado



Fonte: Elaboração própria.

O *bitmap* consiste em uma matriz de pixels, no caso do trabalho 8x4 pixels, em que o estado do pixel define a sua cor e posição na imagem em jogo, o que cria uma forma que pode ser acessada diretamente, o que evita que se tenha que desenhar pixel a pixel os *sprites*[10] do jogo.

5.2.6 INTERRUPÇÕES

Quando um dispositivo precisa da atenção da CPU, ele gera uma interrupção, que é basicamente um desvio na execução normal da sequência de instruções do processador, ou seja, a interrupção requisita atenção da CPU para algum dispositivo.

Após concluir essa rotina, a CPU volta a realizar a tarefa que estava executando antes de ser interrompida. Entrar no modo de vídeo, pressionar uma tecla e reproduzir um som, são exemplos de interrupções (HYDE, 2010).

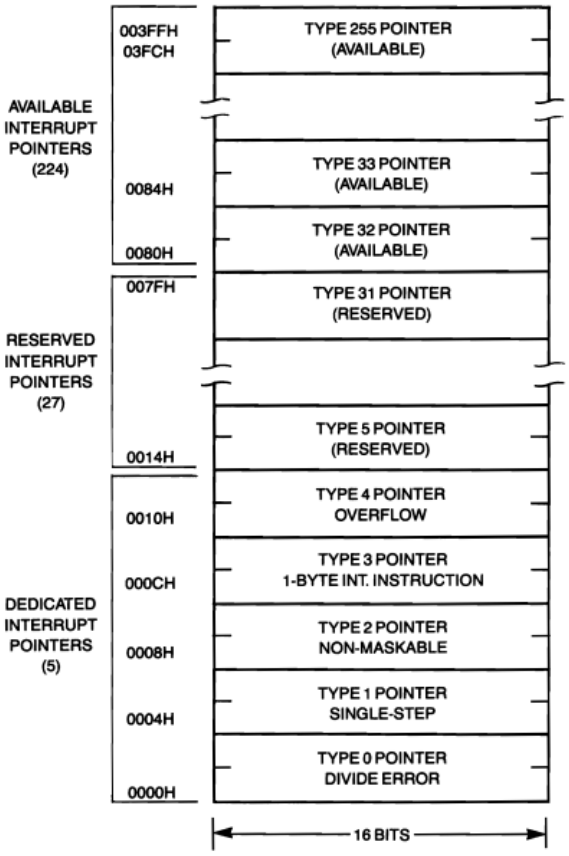
Nos processadores 8086/88, é recebido um sinal de interrupção, que pode ser interno, através de uma instrução “INT”, ou externo, por meio de um pedido de *hardware*. Com o reconhecimento do sinal de interrupção, o processador salva o estado atual do programa (IP, PSW) na pilha, consulta o vetor de interrupção para encontrar o endereço das rotinas de serviço de interrupção (ISRs) na memória, carrega este endereço a partir do vetor requisitado e começa a executá-lo.

Após a conclusão da rotina (ISR), uma instrução “IRET” é usada para retornar da interrupção, o que restaura o estado do programa que havia sido salvo, portanto o programa continua de onde parou antes da instrução ser executada. Cada tipo de interrupção requer 4 bytes de memória e precede de um endereço específico localizado nos primeiros 1024 bytes da memória, de 0000:0000 a 0000:03FF, esta faixa de endereços pode ser observada na Figura 23 (COFFRON, 1983).

Por exemplo, para executar a interrupção para ativar o modo de manipulação de vídeo, é necessário usar a função INT seguida por um valor que indica qual interrupção deve ser executada, neste caso, o valor é 10h. No caso da interrupção de manipulação de vídeo, há subfunções, definidas pelo valor contido no registrador AX.

[10]Descreve uma imagem ou animação em duas dimensões que é incorporada em um cenário de jogo mais amplo. Normalmente, os *sprites* são empregados para ilustrar personagens, objetos e diversos efeitos visuais em jogos (LENOVO, 2023).

Figura 23 - Interrupções do 8086/88.



Fonte: (COFFRON, 1983)

Para ativar o modo de vídeo 320x200 pixels e 256 cores, o byte mais significativo do registrador AX deve ser 00h, que requisita que a interrupção defina o modo de vídeo, e dentro dos modos de vídeos disponíveis para o 8086, o modo VGA mencionado possui o valor 13h, que deve ser o valor do byte menos significativo do registrador AX (COFFRON, 1983). Dessa forma a interrupção para ativar o modo de vídeo de interesse foi feita do seguinte modo:

$mov\ ax, 0013h$

$int\ 10h$

A interrupção INT 10h é utilizada em aplicações que requerem controle direto sobre exibição de vídeo, seja em modo texto, ou gráfico.

5.2.7 INSTRUÇÕES DE CADEIAS DE CARACTERES (STRINGS)

A arquitetura x86 possui doze instruções de cadeias de caracteres, que podem ser usadas para manipular *strings* (*byte*, *word* ou *double word*[11]), essas instruções são essenciais para a manipulação eficiente de dados em memória. No contexto deste trabalho, destaca-se o uso das instruções *lodsb* e *stosb*. A instrução *lodsb* é utilizada para carregar um elemento de *string* (*byte*) do endereço apontado por DS:SI no registrador AL, enquanto *stosb* armazena o conteúdo do registrador AL no endereço apontado por ES:DI, ambas instruções também incrementam um valor relativo ao tamanho da *string* após sua execução. Essas operações são cruciais para tarefas que envolvem a leitura e escrita de dados em locais específicos da memória, como por exemplo a memória de vídeo (HYDE, 2010).

As instruções de *string* podem ser potencializadas quando combinadas com prefixos como *rep*, *repz*, repe, *repnz* e *repne*, permitindo o processamento automatizado de uma *string* inteira. As instruções rep e *loop* utilizam o registrador de uso geral cx como contador, o decrementando toda iteração até cx chegar a zero. Essa capacidade torna a arquitetura x86 poderosa para tarefas de processamento de dados e manipulação de *strings* em baixo nível (HYDE, 2010).

5.2.8 INSTRUÇÕES DA PILHA

A pilha funciona como uma estrutura do tipo LIFO (*Last-In, First-Out*), característica que define sua operação de armazenamento e recuperação de dados. Em resumo, o último elemento a ser inserido na pilha é sempre o primeiro a ser retirado. O que o torna útil em cenários como chamadas de subrotinas e manipulação de estados temporários, onde a ordem de execução e recuperação dos dados é crucial (COFFRON, 1983).

O funcionamento da pilha é intimamente ligado aos registradores SP (*Stack* Pointer) e ao segmento de pilha SS (*Stack Segment*). O registrador SP desempenha um papel crucial no rastreamento do 'topo' da pilha. Ao realizar uma operação de PUSH, o SP é decrementado em 2, refletindo a adição de uma palavra de dados (2 bytes) na pilha, e o dado é armazenado no endereço de memória correspondente. Inversamente, a operação de POP resulta no incremento de SP em 2 após a leitura de uma palavra da pilha, preparando-a para o próximo

[11]A palavra dupla, ou em inglês *double word*, está disponível apenas do processador 80386 adiante.

acesso. As instruções CALL e RET também dependem do SP. Dessa forma a pilha é mantida em seu estado correto, assegurando que os dados mais recentes estejam sempre acessíveis no topo da pilha e que o espaço seja liberado de forma apropriada após a remoção dos dados (HYDE, 2010).

5.2.9 *INSTRUÇÕES CALL E RET.*

As instruções CALL e RET contribuíram para a eficiência e organização do código. A instrução CALL permitiu a organização das múltiplas sub-rotinas, o que dividiu o programa em blocos menores e mais fáceis de gerenciar. Ao utilizar CALL, as próximas instruções para o retorno foram armazenadas na pilha, o que permitiu que após a instrução RET, presente na sub-rotina, o ponteiro de instruções retornasse para o valor aa instrução imediatamente após a instrução de CALL realizada.

5.2.10 *RÓTULOS DE INSTRUÇÃO*

Os rótulos de instrução, conhecidos como "*statement labels*" (o rótulo seguido de dois pontos), foram utilizados para a eficiência e clareza do código. Estes rótulos, que atuam como marcadores no fluxo do programa, permitiram a definição dos pontos de retorno e salto de maneira precisa e organizada.

Utilizar um rótulo de instrução, como por exemplo *"rotinaprincipal:*", proporcionou a capacidade de referenciar esse ponto específico no código diretamente em instrução de salto, como *"jmp rotinaprincipal*"". Essa prática eliminou a complexidade de calcular manualmente os endereços de destino das instruções.

5.2.11 *INICIALIZAÇÃO E CONFIGURAÇÃO DE VÍDEO*

A inicialização do programa consistiu em configurar a máquina como um setor de inicialização e preparar as instruções iniciais para entrar na rotina principal do jogo. O programa foi iniciado como um setor de inicialização, foram definidas as variáveis, constantes e o modo gráfico.

5.2.12 *ROTINA PRINCIPAL*

A rotina principal utiliza as instruções CALL e RET e executa sequencialmente as subrotinas que foram criadas ao longo do programa. Assim, para futuros aprimoramentos, poderiam ser feitas adaptações e novas funcionalidades de maneira que não prejudicasse a visualização e compreensão das rotinas que operam no jogo. A rotina `checarteclado` foi responsável pela verificação das entradas do teclado ‘d’ e ‘a’, o que permitiu a movimentação da nave, para a direita, ao incrementar o valor de navex, ou esquerda, ao decrementar o valor de navex, endereço de memória utilizado na rotina de posição da nave, enquanto a `limpartela` cuidou da atualização da tela a cada quadro do jogo, o que assegurou que a posição antiga da nave fosse apagada antes de ser desenhada na posição atualizada. A `posnave` calculou a posição atual da nave, com base no valor modificável do endereço de memória navex e no valor fixo da constante navey, e carregou no registrador apontador de destino (DI), que foi utilizado, logo após, pela rotina `desenharnave` que foi encarregada de desenhar a nave na posição calculada, ao ler seu *bitmap* e atualizar na tela pixel a pixel. Por fim, a sub-rotina *`delay`*, que introduziu uma pausa no *loop* principal, o que permitiu certo controle na velocidade do jogo e, consequentemente, na atualização da posição da nave.

Este programa representou um exemplo prático do uso da linguagem de montagem x86 para criar um jogo interativo em um ambiente de setor de inicialização, o que demonstrou o controle de hardware em baixo nível e a manipulação direta de memória e dispositivos de entrada. As habilidades e técnicas apresentadas neste projeto foram fundamentais para entender os princípios da computação e do desenvolvimento de jogos em nível de sistema.

5.3 JOGO EM EXECUÇÃO

Na Figura 24 é apresentada a imagem do jogo em funcionamento, via emulador QEMU. O jogo possui entrada para 2 movimentos na horizontal, se mover para a esquerda ao pressionar a tecla “a”, ou se mover para a direita ao pressionar a tecla “d”. Todos os mnemônicos do jogo encontram-se no Apêndice B.

Figura 24 – Captura de tela do jogo em funcionamento.

Fonte: Elaboração própria.

Observa-se pela figura 18, que dos 512 bytes que havia para a execução do jogo, utilizou-se 177 bytes, consequentemente restando 335 bytes disponíveis para implementações, além da possibilidade de otimizar o código já feito, reduzindo ainda mais seu consumo.

6 CONCLUSÃO

O trabalho de graduação proporcionou um aprofundamento nas linguagens de montagem e em sua aplicabilidade no desenvolvimento de jogos digitais, um campo que, embora complexo, revelou-se extremamente enriquecedor. Foi explorada não apenas a compreensão técnica, mas também o entendimento do papel cultural e histórico dos jogos, principalmente dos jogos digitais.

Através deste estudo, foi possível entender a interação das linguagens de montagem com as arquiteturas de CPU, destacando-se a influência significativa da arquitetura de Von Neumann. O trabalho proporcionou uma perspectiva histórica, ilustrando como as mudanças tecnológicas moldaram a indústria de jogos digitais, especialmente sob a luz dos avanços recentes e do impacto da pandemia do coronavírus SARS-CoV-2.

Um dos aspectos mais desafiadores, mas igualmente gratificantes, foi o desenvolvimento elementos de um jogo em linguagem de montagem x86. Esse processo não apenas aplicou os conhecimentos teóricos adquiridos, mas também forneceu percepções valiosas sobre as complexidades do *design* de jogos. A implementação experimental foi um teste da interconexão entre a programação e a criatividade no design de jogos.

A experiência adquirida neste estudo serve como um ponto de partida para aprofundamentos, tanto na linguagem de montagem quanto no vasto campo do desenvolvimento de jogos.

REFERÊNCIAS

AFTERLIFE. **Interação e gráficos: Games antes dos anos 2000**. Medium, 2016. Disponível em: https://medium.com/@afterlife/intera%C3%A7%C3%A3o-e-gr%C3%A1ficos-games-antes-dos-anos-2000-a1027a9d07d8. Acesso em: 01 nov. 2023.

ALFIUNE, Pepita de Souza; CUSTÓDIO, José Antônio Loures. **O Ethos religioso na antiguidade: a origem ritualística dos jogos de tabuleiro**. FAP: Revista de artes do campus Curitiba II, 2019. Disponível em: https://periodicos.unespar.edu.br/index.php/revistacientifica/article/view/2480.

ALVES, Marcus Vinícius. **Estudo de técnicas de nanofabricação aplicadas a filmes semicondutores**. Dissertação apresentada ao Instituto de Física de São Carlos, Universidade de São Paulo, para obtenção do título de Mestre em Ciências-Física Aplicada, 1999. Disponível em: https://www.teses.usp.br/teses/disponiveis/76/76132/tde-01042014-171353/publico/MarcusAlvesM.pdf.

ANDRADE, Nilson S. de; NETO, A.V. Andrade; LEMAIRE, Thierry; CRUZ, J.A. **Investigação teórica e experimental do efeito termiônico**. Artigos Gerais, 2013. Disponível em: https://doi.org/10.1590/S1806-11172013000100008.

ANONYMOUS. **Uso das válvulas nos computadores**. Trabalhos física vet, 2012. Disponível em: http://trabalhofisicavet.blogspot.com/2012/09/uso-das-valvulas-nos-computadores.html. Acesso em: 10 nov. 2023.

ASHBURN, Doug. **Bell Laboratories**. The Editors of Encyclopaedia Britannica, 2023. Disponível em: https://www.britannica.com/topic/Bell-Laboratories.

BELLARD, Fabrice. (2005). **QEMU, a Fast and Portable Dynamic Translator**. 41-46.

BLUNDELL, Nick. **Writing a Simple Operating System from Scratch.** United Kingdom: University of Birmingham. 2010.

Board Game Rank. Board Game Geek. Disponível em: https://boardgamegeek.com/browse/boardgame. Acesso em: 14 nov. 2023.

BRAGA, Newton C. **A Lei de Moore.** Instituto NCB. [s.d.]. Disponível em: https://www.newtoncbraga.com.br/projetos/8084-a-lei-de-moore-art1177.html. Acesso em: 14 nov. 2023.

BRAGA, Newton C. **Como é fabricado um circuito integrado**. Instituto NCB. [s.d.]. Disponível em: https://www.newtoncbraga.com.br/como-funciona/18214-como-e-fabricado-um-circuito-integrado-art4557.html. Acesso em: 14 nov. 2023.

BRANDÃO, Alexandro. **Videogame Atari 2600**. Disponível em: https://sites.unoeste.br/museu/Atari-2600/.

CARBONE, Filipe. **Starfield ocupará 125 GB no Xbox**. Adrenaline, 2023. Disponível em: https://www.adrenaline.com.br/games/starfield-ocupara-125gb-xbox/.

CARDI, Marilza de Lourdes. **Evolução da computação no Brasil e sua relação com fatos internacionais.** Dissertação submetida à Universidade Federal de Santa Catarina como parte dos requisitos para obtenção do grau de Mestre em Ciência da Computação, 2002.

CARDOSO, Pollyana Costa; CALDEIRA, Thaís Cristina Marquezine; SOUSA, Taciana Maia De; CLARO, Rafael Moreira. **Changes in Screen Time in Brazil: A Time-Series Analysis 2016-2021.** American Journal of Health Promotion, 2023. Disponível em: https://journals.sagepub.com/doi/10.1177/08901171231152147.

COFFRON, James. **Programming the 8086/8088**. 1ª ed. Sybex. Berkeley, California. 1983.

CRAWFORD, Chris. **The art of Computer Game Design: Reflections of a Master Game Design**. McGraw-Hill Osborne Media, 1984.

CSIKSZENTMIHALYI, Mihaly. **Play and Intrinsic Rewards**. Journal of Humanistic Psychology, 1975.

EDITORA, Porto. **UNIVAC na Infopédia**. Porto Editora. Disponível em: https://www.infopedia.pt/$univa. Acesso em: 10 nov. 2023.

FELDMAN, Martin. **Nanolithography: The Art of Fabricating Nanoelectronic and Nanophotonic Devices and Systems**. Woodhead Publishing Series in Electronic and Optical Materials Book 42, 2014.

FROUGIER, Julien; GUO, Dechao. **Introducing the world's first 2 nm node chip**. Research IBM, 2021. Disponível em: https://research.ibm.com/blog/2-nm-chip.

GNU Make – Version 4.3, Free Software Foundation, 2020. Disponível em: https://www.gnu.org/software/make. Acesso em: 11 nov. 2023.

GUTIÉRREZ, Óscar. **Programming Boot Sector Games.** 1ª ed. Lulu Press. 2019.

GROUP, Sioux; GAMERS, Go. **Report gratuito BR v2.2-final**. Pesquisa Game Brasil, 2023.

GROUP, Sioux; GAMERS, Go. **Report gratuito BR-final**. Pesquisa Game Brasil, 2021.

HERMAN, Leonard. **Phoenix: The Fall & Rise of Videogames**. Rolenta Press, 2001.

HYDE, Randall. **The art of assembly language.** 2ª ed. No Starch Press. 2010.

HUIZINGA, Johan. **Home Ludens: A study of the Play Element in Culture**. Beacon Press, 2014:1971.

IBUKA, Masaru. **Masaru Ibuka**. Sony, 1998. Disponível em: https://www.sony.com/en/SonyInfo/News/Press/199801/ibuka-e.html.

IMARC. **Game based learning market**. 2022. Disponível em: https://www.imarcgroup.com/game-based-learning-market. Acesso em:

JEON, Sangeon; CHO, Jaewan; CHO, Sung Min. **Photolithography-free fabrication of a-IGZO thin film transistor with interconnecting metal lines.** Materials Science in Semiconductor Processing, 2021 Disponível em: https://doi.org/10.1016/j.mssp.2020.105417.

JUUL, Jesper. **Half-Real: Video Games between Real Rules and Fictional Worlds**. Mit Press, 2011.

KELLEHER, Dr. Ann. **Lei de Moore – Agora e no Futuro**. Intel, 2022. Disponível em: https://www.intel.com.br/content/www/br/pt/newsroom/opinion/moore-law-now-and-in-the-future.html.

LENOVO. **What is a sprite?** Disponível em: https://www.lenovo.com/us/en/glossary/sprite/?orgRef=https%253A%252F%252Fwww.google.com%252F. Acesso em: 08 nov. 2023.

MIRANDA, Simão de. **No Fascínio do jogo, a alegria de aprender**. Revista semestral da Faculdade de Educação, 2002. Disponível em: https://periodicos.unb.br/index.php/linhascriticas/article/view/2989.

MURR, Caroline; FERRARI, Gabriel. **Entendendo e aplicando a gamificação, o que é, para que serve, e desafios**. Florianópolis: UFSC: UAB. 2020

NASM – The Netwide Assembler. Version 2.15.05, the NASM Authors, 2021. Disponível em: https://www.nasm.us/index.php. Acesso em: 11 nov. 2023.

NEWZOO. **Newzoos's Global Games Market Report 2023**. Free Version, 2023. Disponível em: https://newzoo.com/resources/trend-reports/newzoo-global-games-market-report-2023-free-version?v=3. Acesso em: 14 nov. 2023.

PALLARDY, Richard. **Sir John Ambrose Fleming**. The Editors of Encyclopedia Britannica, 2023. Disponível em: https://www.britannica.com/biography/John-Ambrose-Fleming.

PARK, William. **A invenção milenar que deu origem aos jogos de tabuleiro atuais**. BBC News Brasil, 2021. Disponível em: https://www.bbc.com/portuguese/vert-fut-56485382.

PIROPO, B. **A invenção do Circuito Integrado**. Techtudo, 2012. Disponível em: https://www.techtudo.com.br/noticias/2012/11/a-invencao-do-circuito-integrado.ghtml.

PIROPO, B. **Os primeiros transistores**. Techtudo, 2012. Disponível em: https://www.techtudo.com.br/noticias/2012/10/os-primeiros-transistores.ghtml.

POITIERS, N; WEIL, P. **A new direction for the European Union's half-hearted semiconductor strategy**. Policy Contribution, 2021, Bruegel. Disponível em: https://www.bruegel.org/sites/default/files/wp_attachments/PC-2021-17-semiconductors-.pdf

REDAÇÃO, Da. **Lista reúne os primeiros jogos lançados no mundo, de SpaceWar a Tank**. Techtudo, 2016. Disponível em: https://www.techtudo.com.br/noticias/2016/01/lista-reune-os-primeiros-jogos-lancados-no-mundo-de-spacewar-tank.ghtml. Acesso em: 14 nov. 2023.

ROCHESTER, Nathaniel. **The 701 Project as Seen by Its Chief Architect**. IEEE, 1983. Disponível em: https://ieeexplore.ieee.org/document/4640454.

ROUSE, Richard; OGDEN, Steve; FALSTEIN, Noah. **Game Design: Theory & Practice, Second Edition.** Wordware Publishing, 2004.

SANTOS, Renato Machado dos. **História da peteca**. CBP: Confederação Brasileira de Peteca, 2020. Disponível em: https://cbpeteca.org.br/historia-da-peteca/.

SEVERANCE, Chuck. **What a Compiler Does.** LibreTexts Engineering. Disponível em: https://eng.libretexts.org/Bookshelves/Computer_Science/Programming_and_Computation_Fundamentals/High_Performance_Computing_(Severance)/03%3A_Programming_and_Tuning_Software/3.01%3A_What_a_Compiler_Does.

SPORTS, Electronic Arts. **FIFA International Soccer**. PlayStation; Xbox; PC, 1993. Jogo eletrônico.

STALLINGS, William. **Arquitetura e Organização de Computadores**. Pearson, 2017.

STEAM. **Visão geral das listas: os jogos mais vendidos e mais jogados do Steam.** Steam Powered, 2023. Disponível em: https://store.steampowered.com/charts/. Acesso em: 10 nov. 2023.

TANENBAUM, Andrew S. **Organização Estruturada de computadores.** Pearson, 2013.

TEKINBAS, Katie Salen; ZIMMERMAN, Eric. **Rules of Play: Game Design Fundamentals**. MIT Press, 2003.

The 8086 Family User's Manual. Disponível em: http://www.righto.com/2020/07/the-intel-8086-processors-registers.html#fnref:multiport. Acesso em: 10 nov. 2023.

VASSAU, Kyle. **Makefile**. Disponível em: https://git.sr.ht/~queso_fuego/bootsector_games/tree/master/item/space_invaders/makefile. Acesso em: 11 nov. 2023.

Video Games – Worldwide. Statista, 2023. Disponível em: https://www.statista.com/outlook/dmo/digital-media/video-games/worldwide#:~:text=In%202023%2C%20the%20Video%20Games,in%20a%20projected%20market%20volume. Acesso em: 10 nov. 2023.

Visual Studio Code. Version 1.84.2, Microsoft Corporation, 2023. Disponível em: https://code.visualstudio.com. Acesso em: 11 nov. 2023.

WELSH, Stephen; KNAGGS, Peter. **ARM: Assembly Language Programming**. Bournemouth University, 2003.

APÊNDICE A – Código-Fonte Do Jogo

```
1   org 7c00h
2
3   navex           equ  7d00h
4   navey           equ  100
5   largura         equ  320
6   altura          equ  200
7   cordanave       equ  0fh
8   escalaaltura    equ  2
9   escalalargura   equ  2
10
11  mov ax, 0013h
12  int 10h
13
14  mov ax, cs
15  mov ds, ax
16  mov ax, 0A000h
17  mov es, ax
18
19  mov ax, 160
20  mov [navex], ax
21
22  rotinaprincipal:
23      call checarteclado
24      call limpartela
25      call posnave
26      call desenharnave
27      call delay
28      jmp rotinaprincipal
29
30  limpartela:
31      pusha
32      xor ax, ax
33      xor di, di
34      mov cx, largura * altura
35      rep stosb
36      popa
37      ret
38
39  posnave:
```

```
40      mov ax, navey
41      mov bx, largura
42      mul bx
43      add ax, [navex]
44      mov di, ax
45      ret
46
47  desenharnave:
48      pusha
49      mov si, navebitmap
50      mov dx, 4
51      .proxlinha:
52          lodsb
53          mov cx, 8
54          .proxpixel:
55              push di
56              mov bh, escalaaltura
57              .linhaPixel:
58                  push cx
59                  mov cx, escalalargura
60                  .pixel:
61                      test al, 80h
62                      jz .skipPixel
63                      mov byte [es:di], cordanave
64                      .skipPixel:
65                      inc di
66                  loop .pixel
67                  pop cx
68                  add di, largura - escalalargura
69                  dec bh
70              jnz .linhaPixel
71              pop di
72              add di, escalalargura
73              shl al, 1
74              dec cx
75          jnz .proxpixel
76          add di, largura * (escalaaltura - 1)
77          sub di, 8 * escalalargura
78          dec dx
79      jnz .proxlinha
```

```
80      popa
81      ret
82
83  delay:
84      pusha
85
86      mov cx, 0EFFFh
87      .loopdelay:
88          loop .loopdelay
89      popa
90      ret
91
92  checarteclado:
93      pusha
94      mov ah, 01h
95      int 16h
96      jz nenhumatecla
97      mov ah, 00h
98      int 16h
99      cmp al, 'a'
100     je moveresquerda
101     cmp al,'d'
102     je moverdireita
103     jmp nenhumatecla
104
105 moveresquerda:
106     sub word [navex], 8
107     jmp fimchecarteclado
108
109 moverdireita:
110     add word [navex], 8
111     jmp fimchecarteclado
112
113 fimchecarteclado:
114 nenhumatecla:
115     popa
116     ret
117
118 navebitmap:
119 db 00011000b
```

```
120 db 01011010b
121 db 01111110b
122 db 00111100b
123
124 times 510 - ($ - $$) db 0
125 db 55h
126 db 0aah
```

APÊNDICE B – MNEMÔNICOS DO 8086/8088 UTILIZADOS NO JOGO

Tabela 1 – Mnemônicos utilizados no código fonte do jogo

Instrução	**Descrição resumida**
ADD	Adição
CALL	Chama uma sub-rotina
CMP	Compara dois operandos
DEC	Decrementa um operando
INC	Incrementa um operando
INT	Chamada para interrupção
JE	Salto se igual
JMP	Salto incondicional
JZ	Salto se zero
LODSB	Carrega byte de DS:SI em AL
MOV	Move dados
MUL	Multiplica AX por um operando
ORG	Define endereço de origem do código
POPA	Desempilha todos os registradores gerais
PUSHA	Empilha todos os registradores gerais
REP	Repete a instrução seguinte [CX] vezes
RET	Retorna de uma sub-rotina
SHL	Desloca os bits para a esquerda
STOSB	Armazena byte de AL em ES:DI
SUB	Subtração
TEST	Teste lógico “AND” entre operandos
XOR	Operação lógica “XOR” entre operandos

Fonte: Elaboração própria