http://journal.unoeste.br/index.php/ce/index
DOI: 10.5747/ce.2024.v16.e417
ISSN on-line 2178-8332

Colloquium Exactarum

Submetido: 20/01/2023 Revisado:10/10/2023 Aceito: 27/02/2024

MODELO SEMÂNTICO DE OPERAÇÕES ARITMÉTICAS E LÓGICAS PARA HARDWARE VIRTUAL

SEMANTIC MODEL OF ARITHMETIC AND LOGICAL OPERATIONS FOR VIRTUAL HARDWARE

Igor Sousa dos Santos Santana[1], César Alberto Bravo Pariente[2]

Universidade Estadual de Santa Cruz – UESC, Ilhéus, BA.
e-mail: isssantana.cic@uesc.br, cabpariente@uesc.br

RESUMO – O objetivo desse projeto foi desenvolver um modelo semântico de operações aritméticas e lógicas para hardware virtual; esse objetivo foi atingido através de três etapas principais: Implementar portas lógicas e unidades lógicas e aritméticas no software simulador de hardware virtual Nand2tetris; Implementar funções recursivas e iterativas, para aferir o correto funcionamento das operações lógicas e aritméticas na máquina virtual p-code machine. Na primeira etapa, foram desenvolvidas no software simulador de hardware Nand2tetris implementações das portas lógicas elementares. A partir dessas portas lógicas, foram construídos circuitos combinacionais e circuitos sequenciais como unidade lógica e aritmética, Meio Somador, Somador Completo, Adição de 16 bits, negação lógica de 16 bits, E lógico de 16 bits, OU lógico de 16 bits, entre outros. Na segunda etapa tratou-se de experimentar em alto nível as operações aritméticas e lógicas, disponibilizadas pela máquina virtual p-code machine, que foi implementada na linguagem de programação C, como soma, subtração, maior que, menor que. Na terceira etapa, com as operações lógicas e aritméticas já implementadas, foram desenvolvidos programas iterativos e recursivos para calcular o *n*-ésimo valor da Sequência de Fibonacci e o fatorial de um número *n* qualquer.
Palavras-chave: Virtualização; Hipervisor; Hardware virtual.

ABSTRACT – The objective of this project was to develop a semantic model of arithmetic and logical operations for virtual hardware; this goal was achieved through three main steps: Implement logic gates and arithmetic and logic units in Nand2tetris virtual hardware simulator software; Implement recursive and iterative functions to check the correct functioning of logical and arithmetic operations in the p-code machine virtual machine. In the first stage, implementations of the elementary logic gates were developed in the hardware simulator software Nand2tetris. From these logic gates, combinational circuits and sequential circuits were built as logical and arithmetic unit, Half Adder, Full Adder, 16-bit Addition, 16-bit logical negation, 16-bit logical AND, 16-bit logical OR, among others. The second stage involved experimenting at a high level with arithmetic and logical operations, provided by the virtual machine p-code machine, which was implemented in the C programming language, as addition, subtraction, greater than, less than. In the third stage, having already implemented logical and arithmetic operations, iterative and recursive programs were developed to calculate the nth value of the Fibonacci Sequence and the factorial of any number n.
Keywords: Virtualization; Hypervisor; Virtual Hardware.

1. INTRODUÇÃO

Virtualização é um recurso muito utilizado para reduzir custos de projetos e tornar possível a realização de testes e experimentos de equipamentos, eventos físicos, químicos, nucleares, entre outros.

Os computadores atuais possuem portas lógicas em seu nível mais baixo, o nível de máquina. As portas lógicas seguem o padrão de arquitetura do montador do equipamento e não podem ser modificadas. Isso torna o estudo de portas lógicas muito complexo.

Logic.ly (LOGIC.LY, 2021) e Nand2Tetris (NAND2TETRIS, 2021) são exemplos de ferramentas de simulação de portas lógicas. Elas permitem o estudo das operações lógicas e aritméticas realizadas por portas lógicas.

Em 1972 foi lançado, pela IBM, o System/370, sucessor da família System/360, e a virtualização foi muito difundida nos anos 1980 por conta do baixo custo da utilização da técnica (Campos, 2022a).

Com base nos conceitos de virtualização, de máquinas virtuais e hipervisores, o objetivo do projeto foi pesquisar modelos semânticos de operações aritméticas para hardware virtual, onde foi pesquisado e implementado uma máquina virtual, em modelo de stack, conhecido como p-code machine, na linguagem de programação C a fim de realizar operações aritméticas e lógicas com ela.

Foram desenvolvidas funções recursivas e iterativas para avaliar a implementação das operações aritméticas e lógicas na p-code machine utilizando a linguagem de programação C, e também projetadas e implementadas as unidades lógicas e aritméticas (de 1, 2, 4, 8, 16, 32 e 64 bits) utilizando o software de simulação de hardware Logic.ly.

Testes automatizados foram construídos para as unidades aritméticas e lógicas, para construir os testes automatizados foram implementados contadores síncronos, utilizando flip-flops do tipo T conectados em série e alimentados por um único clock.

2. FUNDAMENTAÇÃO TEÓRICA

Este capítulo define o que é virtualização, hipervisor e hardware virtual e os componentes necessários para sua utilização, apresenta trabalhos relacionados ao projeto desenvolvido, e o estado da arte de máquinas virtuais.

2.1. Virtualização

Segundo Campos (2022a, p. 15) “A virtualização teve seu início nos anos 1960, na época os mainframes eram computadores com muitos recursos e grande velocidade de processamento que, por muitas vezes eram mantidos ociosos, pois os processos eram gerenciados manualmente pelo operador da máquina [...]”.

A IBM foi a primeira empresa a implementar virtualização em seus computadores, com a família System/360, que permitiam, simultaneamente, a execução de diversos sistemas operacionais, além de conseguirem se comunicar (IBM, 2023).

No início da popularização da arquitetura x86 os sistemas operacionais Windows e Linux a virtualização não era suportada. A virtualização foi introduzida à arquitetura x86 pela (VMware, 2023) e a partir de 2005 os fabricantes melhoram o suporte via hardware (HIALINX, 2019).

2.2. Hipervisor

O hipervisor é uma camada de abstração responsável por permitir a hospedagem de VMs, controlar como os recursos da máquina são acessados pelas VMs e alocar dinamicamente os recursos de computação para cada (Campos, 2022a). Os recursos da máquina hospedeira podem ser compartilhados entre as VMs (Singh, 2019).

2.3. Máquina Virtual

“Máquina virtual é um container de software, que possui seu próprio sistema operacional e aplicativos além de ser autônoma” (Zanoello, 2017, p. 9).

Um dos mais comuns usos de virtualização é o de máquinas virtuais (VMs). As mais conhecidas são Oracle Java Virtual Machine (JVM), Microsoft Common Language

Runtime (CLR) e VMware.

De acordo com (Singh, 2019) as máquinas virtuais são criadas para executar tarefas específicas que podem ser prejudiciais ao computador hospedeiro.

A máquina virtual é separada do sistema hospedeiro. Caso ocorra algum erro ou dano na máquina virtual, o mesmo não ocorrerá com o computador físico.

2.3. Trabalhos Relacionados

Campos (2022a) propôs o desenvolvimento de uma máquina virtual, AllanaVM (Campos, 2022b), com base nos conceitos de arquitetura de Brookshear (2005), Von-Neumann e p-code (Wirth, 1996).

O modelo de controle de fluxo e a chamada de subrotinas da AllanaVM foi implementado com base no p-code.

O conjunto de instruções foi estendido da máquina de Brookshear (2005) em conjunto às instruções lógicas e aritméticas da máquina p-code, pois segundo Campos (2022a) o conjunto de instruções da máquina de Brookshear não possui as instruções necessárias para chamadas e retornos de sub-rotinas.

2.4. Estado da Arte

A Oracle possui a Oracle VM, um ambiente equipado para virtualização que permite a implantação de sistemas operacionais e softwares em um ambiente virtualizado e não necessita de um sistema operacional para ser executado (ORACLE, 2021).

A empresa VMware desenvolveu a VMware Workstation Pro, um hipervisor que executa sistemas operacionais Windows e Linux com processadores AMD ou Intel (VMWARE, 2023).

A Máquina Virtual Java (JVM) é um programa que carrega e executa os aplicativos Java, convertendo os bytecodes em código executável de máquina. Programas escritos em Java podem funcionar em qualquer plataforma de hardware e software que possua uma versão da JVM (WIKIPEDIA, 2023).

Microsoft Azure é uma plataforma que oferece serviços em nuvem. Ela permite a criação de máquinas virtuais, softwares, sites e sistemas de gerenciamento de banco de dados (MICROSOFT, 2023).

3. METODOLOGIA

Esta seção define os materiais que foram utilizados durante a implementação do projeto e descreve os métodos aplicados sobre as etapas.

3.1. Material

Os materiais utilizados para realizar o desenvolvimento do projeto foram: o Compilador virtual para Linguagem de Programação C, cpp.sh, a Linguagem de Programação C e o P-code machine implementada em Linguagem C, rodando em cpp.sh e o Software simulador de circuitos lógicos, Logic.ly.

3.2. Procedimentos

Primeiro foi implementado o p-code machine em linguagem C. A implementação da máquina p-code com as principais funções de estrutura de dados em forma de pilha.

Pilhas têm como suas principais funções *PUSH*, que adiciona uma tarefa no topo da pilha, e *POP*, que remove uma tarefa do topo da pilha. É com a função *POP* que a tarefa removida será executada após sua exclusão do topo da pilha.

Com o programa p-code machine completo, foram desenvolvidos códigos para cálculo da Sequência de Fibonacci, iterativo e recursivo, e códigos que calculam o fatorial de um número, recursivamente e iterativamente.

Após a finalização das implementações foi possível aferir o correto funcionamento das operações lógicas e aritméticas. Finalizadas as implementações, foi iniciada a construção das unidades lógicas e aritméticas (de 1, 2, 4, 8, 16, 32 e 64 bits), no software Logic.ly, somente para após sua construção aferir o correto funcionamento das operações lógicas e aritméticas a nível lógico.

Um passo antes de conferir as operações lógicas e aritméticas das unidades lógicas e aritméticas foi implementar o contador síncrono de 4 bits, também no software Logic.ly. Construído com um único clock e quatro flip-flops tipo T, o contador síncrono de 4 bits foi ligado a uma unidade lógica e aritmética de 4 bits, onde o correto funcionamento das operações lógicas e aritméticas foi aferido.

O circuito da ULA possui cinco subcircuitos integrados entre si. Esses subcircuitos são um Gerador Y, um Gerador C, um Multiplexador de duas entradas, um Somador Completo e um Multiplexador de quatro entradas. O Multiplexador de quatro entradas é o operador lógico, ou seja, ele que efetua as operações lógicas como Negação, OU Exclusivo, Disjunção e Conjunção para realizar as operações de comparação como maior que, menor que, igualdade etc. O Gerador Y é o seletor de operações aritméticas, ele depende diretamente do valor do seletor B e dos valores de entrada de S0 e S1. O Gerador C é o bloco que, a partir das entradas de S0 e S1, gera um CARRY IN, ou seja, o "excesso" da operação anterior, todavia o gerador C só é conectado à primeira unidade lógica e aritmética, caso haja mais que uma. O Somador Completo realiza as operações de adição, subtração, incremento e decremento dependendo do valor de saída do subcircuito Gerador Y, ele possui duas saídas. Uma das saídas integra o circuito com o Multiplexador de duas entradas, a segunda saída simula o transporte do resto da iteração atual para a próxima iteração. O Multiplexador de duas entradas une as saídas dos Somador Completo e o Multiplexador de quatro entradas que realiza as operações lógicas e as válida numa única saída com o auxílio da entrada S2.

Contadores são circuitos digitais que variam seus estados, sob o comando de um clock, de acordo com uma sequência predeterminada. São utilizados principalmente para contagens diversas, divisão de frequência, medição de frequência e tempo, geração de formas de onda e conversão de analógico para digital (Capuano; Idoeta, 1997). Existem dois tipos de contadores: os contadores síncronos e os contadores assíncronos. Neste teste, foi utilizado um contador síncrono.

4. RESULTADOS

Os resultados do trabalho foram obtidos por meio de testes automatizados e não automatizados. Os códigos e circuitos construídos para os softwares Logic.ly e P-code machine foram testados de forma padronizada e automatizada. Alguns testes foram nativos do software, já os outros softwares requisitaram que os testes fossem construídos e que suas validações fossem verificadas para, então, serem utilizados nos circuitos e programas.

Os programas construídos para rodar no P-code machine foram verificados a partir de sua saída. As saídas foram verificadas manualmente após a execução do programa, para cada um dos códigos. Os códigos 4 a 14 demonstram o funcionamento da máquina p-code para a sequência de Fibonacci iterativo e recursivo e para cálculo de fatorial iterativo e recursivo.

4.1. P-code machine

Na programação de computadores, uma máquina P-code (portable code machine, em inglês) é uma máquina virtual projetada para executar o p-code, a linguagem assembly ou código de máquina de uma Unidade de Processamento Central (UPC) hipotética (Wirth, 1996).

Os elementos da máquina p-code são compostos por 3 registradores e um conjunto com 8 instruções.

Os programas que foram traduzidos para p-code podem ser interpretados por um programa de software que emulam o comportamento de uma UPC hipotética ou traduzidos para o código de máquina para a UPC na qual o programa deve ser executado e, somente então, o programa será executado.

As seguintes convenções utilizadas no código: l e a são os buffers de nível de endereço, respectivamente. p, b, t são registradores de programa, base e topo da pilha respectivamente. O vetor s é um registrador de instruções e o vetor code é o vetor que utilizamos para preencher com instruções e transmitirmos para o vetor s. Por fim, LIT, OPR, LOD, STO, CAL, INT, JMP e JPC como operações lógicas e aritméticas, como soma, subtração e funções lógicas, como menor que e maior que.

Cada uma das operações realiza uma instrução diferente. LIT é a instrução que usamos para carregar um nova constante na pilha, OPR é a instrução para executar operações matemáticas e lógicas, com ela é possível fazer 13 operações entre elas 5 operações matemáticas, como soma e subtração e 7 comparações lógicas, como igualdade e maior que, LOD é a instrução usada para carregar o valor de uma variável especificada na memória, STO é a instrução usada para atribuir um valor que está na memória a uma variável especificada, CAL é a instrução usada para iniciar uma procedimento em uma posição específica da memória, INT é a instrução usada para incrementar o topo da pilha por um valor específico, ou seja, deslocar qual endereço de memória será utilizado por um valor específico, JMP é a instrução usada

para pular até a instrução especificada, podendo gerar um loop e JPC é a instrução usada para pular uma condicional para uma instrução especificada, também pode gerar um loop.

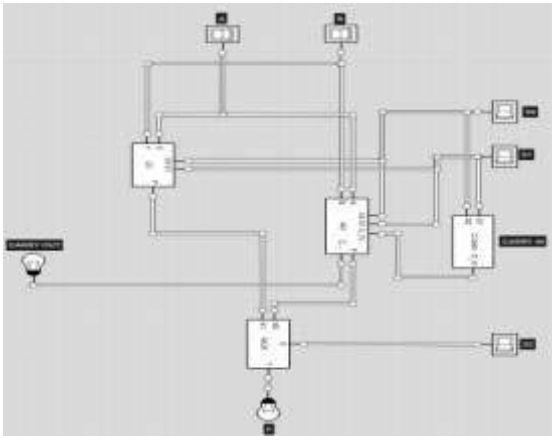
Os programas podem ser modificados na linguagem da máquina P-code sem a necessidade de modificar o código em linguagem de programação C. Para isso é necessário que ele entenda sobre as 8 instruções básicas e utilizá-las como parâmetros para a função que interpreta as instruções. Com a combinação correta das instruções, é possível simular o funcionamento de códigos simples como fatorial de um número, somatório, a Sequência de Fibonacci, etc.

4.2. Teste automatizado para uma ULA de 4 bits e três seletores no software Logic.ly

O experimento utilizou a ferramenta de software Logic.ly (LOGIC.LY, 2021) a fim de ser observado a Unidade Lógica e Aritmética (ULA) como um circuito elétrico com portas lógicas e seus comportamentos distintos. Com a finalidade de maior nível de imersão e aprendizado foi realizado um experimento no software logic.ly, onde foi simulado e observado o comportamento da ULA como um circuito elétrico. A figura 1 exemplifica um circuito ULA desenvolvido no software Logic.ly, seguindo o modelo do livro Digital Logic Circuit Analysis and Design (Nelson *et al*., 1995).

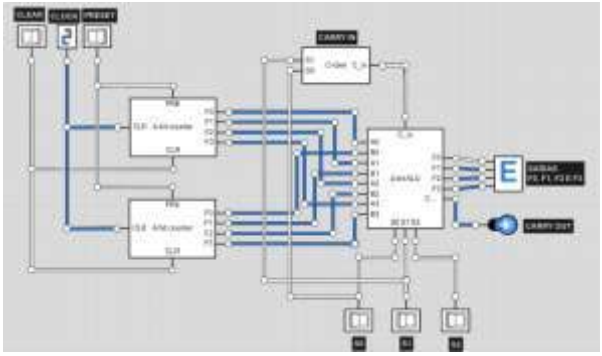
Para o software Logic.ly, foram utilizados tabelas verdades e contadores síncronos, retirados a partir do livro Elementos de Eletrônica Digital (Capuano; Idoeta, 1997) e Digital Logic Circuit Analysis and Design (Nelson, *et al.,* 1995). Os contadores síncronos auxiliaram e automatizaram os testes, de forma fluida e concisa. As figuras 1, 2 e 3 exemplificam a unidade lógica e aritmética de 4 bits, o circuito do teste automatizado e o circuito do contador síncrono.

Figura 1. Circuito ULA de 4 bits e dois seletores no software



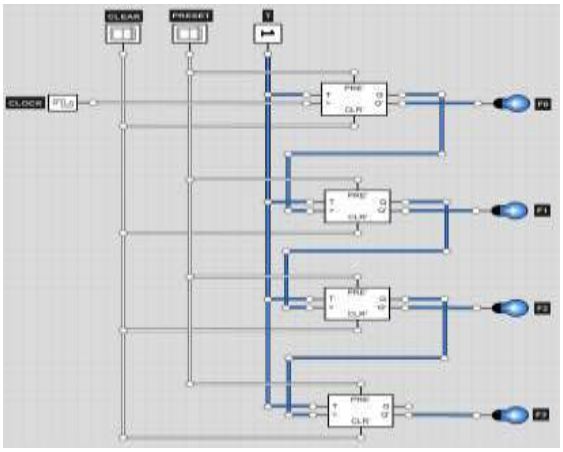
Fonte: Os autores.

Figura 2. Circuito de teste automatizado de uma ULA de 4 bits e 3 seletores.



Fonte: Os autores.

Figura 3. Circuito de contador síncrono de 4 bits.



Fonte: Os autores.

A ULA é um dos mais importantes componentes de um computador. Sem ela, não é possível realizar nenhuma operação lógica ou aritmética, como soma e igualdade, por exemplo. A ULA se encontra onde conhecemos por processador. Ela é a responsável pelo processamento dos dados e informações que vemos através da tela do computador, usamos em compatibilidade com algum outro software e modificamos por meio de um editor de texto ou editor de imagem, por exemplo.

Com o auxílio do software Logic.ly foi possível esquematizar um circuito para uma ULA com as portas lógicas NOT, AND, OR, NAND e XOR. Que são, respectivamente, a negação lógica, a disjunção lógica, a conjunção lógica, a negação da conjunção lógica e a porta lógica conhecida por OU Exclusivo.

Com 6 entradas e 2 saídas, a proposta do circuito foi simular, por meio de pulsos elétricos, o funcionamento de uma ULA. As entradas são: S0, S1, S2, A, B e C_e. As saídas são: C_s e F. Das entradas, apenas S0, S1 e S2 são consideradas entradas de nova informação, A e B são considerados seletores, ou seja, eles controlam o fluxo de informações e seleciona qual será a operação lógica ou matemática que irá acontecer entre as informações de entrada. C_i é o seletor que simula a entrada de um resto da operação anterior, por exemplo, 9 + 1 = 10, o 0 continua na iteração atual, mas a dezena 1 é passada para a próxima iteração, ou seja, para ser somado com o resultado da soma entre os números a sua direita. Foi realizada uma simulação, por meio de vídeo, do circuito da ULA no logic.ly. Foi desenvolvido um vídeo demonstrando o comportamento do circuito para uma ULA no software Logic.ly (Santana, 2021a).

O teste foi realizado com dois contadores síncronos de quatro bits cada, um para o bit de entrada A (A0, A1, A2 e A3) e outro para o bit de entrada B (B0, B1, B2, B3), conectados às suas respectivas entradas de uma unidade lógica e aritmética de 4 bits que estava conectada a uma saída digital de 4 bits. A saída digital de 4 bits representa em hexadecimal (0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E e F) a soma dos oito bits (quatro para A e quatro para B) que recebe em suas entradas.

Deve-se salientar que, como os contadores estavam conectados ao mesmo clock, suas saídas eram idênticas. Portanto, todas as operações lógicas e aritméticas foram conferidas. Soma, subtração, incremento em 1, decremento em 1, E lógico, OU lógico, NÃO lógico do bit A e OU EXCLUSIVO lógico. As saídas para a subtração e para o OU EXCLUSIVO lógico foram nulas, pois as entradas A e B eram idênticas. A figura 2 exemplifica o teste automatizado da unidade lógica e aritmética de 4 bits. Foi desenvolvido um vídeo demonstrando o teste automatizado para a unidade lógica e aritmética de 4 bits (Santana, 2021b).

5. DISCUSSÃO

O primeiro programa feito foi o fatorial de 5, onde o simulador foi especificamente ajustado para aquela tarefa (Santana, 2020). O programa que calcula a Sequência de Fibonacci iterativo (Santana, 2021c) pode calcular qualquer elemento da sequência com a modificação de apenas um único valor, encontrado na linha 215 do código fonte.

O programa que calcula a Sequência de Fibonacci de forma recursiva atua exatamente igual ao programa que calcula de forma iterativa, mas com poucos diferenciais. O cálculo recursivo demanda de mais tempo para ser compilado e executado, além de necessitar de mais trechos e linhas de código para

funcionar corretamente.

O código de fatorial recursivo foi construído a partir dos conceitos do código da sequência de Fibonacci iterativo e as instruções do fatorial iterativo.

Em 100% dos testes realizados com a máquina p-code foi possível observar diferenças de tamanhos de pilhas de execução de algoritmos iterativos e algoritmos recursivos.

As pilhas resultantes da execução de algoritmos iterativos eram menores, ou seja, empilhavam menos instruções e dados, e percorriam menos etapas para a execução das tarefas em comparação com os algoritmos recursivos construídos para a execução das mesmas tarefas. Do ponto de vista computacional, os códigos iterativos consomem menos recursos da plataforma, e desempenham tarefas em tempos menores em relação aos mesmo código implementados de forma recursiva. Isto é uma característica da implementação recursiva de algoritmos. Gasta-se mais recursos da plataforma.

6. CONSIDERAÇÕES FINAIS

Os modelos semânticos construídos com operações lógicas e aritméticas por meio de implementações de funções iterativas e recursivas, a implementação de unidades lógicas e aritméticas foram de grande importância para o entendimento de um modelo semântico de operações aritméticas e lógicas para hardware virtual que são baseadas nos conceitos de arquitetura de Brookshear, Von-Neumann e p-code.

7. APÊNDICE

7.1. Explicação do funcionamento de Sequência de Fibonacci iterativo

O programa começa seu funcionamento com a linha 197, com a instrução code[0].f = INT;code[0].l = 0;code[0].a = 6;. Com essas instruções, são reservadas 6 posições de memória na nossa pilha, sendo 3 posições para o sistema do próprio p-code e 3 posições para o usuário. Na linha 198 cria-se o acumulador de FIB e inicializa-a com 1, com a linha de código: code[1].f = LIT; code[1].l = 0;code[1].a = 1;. Na linha 199 armazena-se a informação que está no topo da pilha na primeira posição livre de memória, com a linha de código: code[2].f = STO; code[2].l = 0;code[2].a = 3;. Na linha 200, cria-se uma variável auxiliar no topo da pilha e inicializa-a com 0, com a linha de código: code[3].f = LIT; code[3].l = 0;code[3].a = 0;. Na linha 201 armazenamos a informação que está no topo da pilha na segunda posição livre de memória, com a linha de código: code[4].f = STO; code[4].l = 0;code[4].a = 4;. Na linha 202 cria-se o contador e o inicializa com 1, com a linha de código: code[5].f = LIT; code[5].l = 0; code[5].a = 1;. Na linha 203, armazenamos a informação que está no topo da pilha na terceira posição livre de memória, com a linha de código: code[5].f = STO; code[5].l = 0; code[5].a = 5;. Na linha 204, carrega-se o contador no topo da pilha, com a linha de código: code[11].f = LOD; code[11].l = 0;code[11].a = 5;. Na linha 205 carrega-se 1 no topo da pilha, com a linha de código: code[8].f = LIT; code[8].l = 0;code[8].a = 1;. Na linha 206 o contador é incrementado pelo número 1 carregado na pilha anteriormente, com a linha de código: code[9].f = OPR; code[9].l = 0; code[9].a = 2;. Na linha 207 armazenamos a informação que está no topo da pilha na terceira posição livre de memória, com a linha de código: code[10].f = STO; code[10].l = 0; code[10].a = 5;. Na linha 208, carrega-se a variável auxiliar no topo da pilha, com a linha de código: code[11].f = LOD; code[11].l = 0;code[11].a = 2;. Na linha 209 carrega-se o acumulador de FIB no topo da pilha, com a linha de código: code[12].f = LOD; code[12].l = 0;code[12].a = 3;. Na linha 210 carrega-se o acumulador de FIB no topo da pilha, com a linha de código: code[13].f = LOD; code[13].l = 0;code[13].a = 3;. Na linha 211, a informação que está localizada no topo da pilha é armazenada na segunda posição de memória, com a linha de código: code[14].f = STO; code[14].l = 0;code[14].a = 4;. Na linha 212, o acumulador de fib(n) é somado com fib(n - 1), com a linha de código: code[15].f = OPR; code[15].l = 0;code[15].a = 2;. Na linha 213, a informação encontrada no topo da pilha é armazenada na primeira posição de memória, com a linha de código: code[16].f = STO; code[16].l = 0;code[16].a = 3;. Na linha 214 o contador é carregado no topo da pilha, com a linha do codigo: code[17].f = LOD; code[17].l = 0;code[17].a = 5;. Na linha 215 o limite é carregado no topo da pilha, com a linha de código: code[18].f = LIT; code[18].l = 0;code[18].a = 10;. Na linha 216 é feita a operação lógica menor que, para o programa saber se pode por finalizar o laço de repetição, com a linha de código: code[19].f = LIT; code[19].l = 0;code[19].a = 10;. Na linha 217 o programa efetua um laço de repetição, retornando o programa para a linha 7 caso o contador seja menor que o limite que foi carregado na pilha com a linha de

código: code[20].f = LIT; code[20].l = 0;code[20].a = 7;, caso o contador seja maior que o limite que foi carregado na pilha, o programa não efetua o laço de repetição. Na linha 218 é efetuado a operação return para evitar que o programa entre em loop infinito, finalizando a execução do código, com a linha de código: code[21].f = OPR; code[21].l = 0; code[21].a = 0;.

7.2. Exemplificação de entradas e saídas da máquina p-code

Código 1. Instruções no vetor code para o cálculo iterativo de Fibonacci de 3.

```
code[ 0].f = INT; code[ 0].l = 0; code[ 0].a = 6;
code[ 1].f = LIT; code[ 1].l = 0; code[ 1].a = 1;
code[ 2].f = STO; code[ 2].l = 0; code[ 2].a = 3;
code[ 3].f = LIT; code[ 3].l = 0; code[ 3].a = 0;
code[ 4].f = STO; code[ 4].l = 0; code[ 4].a = 4;
code[ 5].f = LIT; code[ 5].l = 0; code[ 5].a = 1;
code[ 6].f = STO; code[ 6].l = 0; code[ 6].a = 5;
code[ 7].f = LOD; code[ 7].l = 0; code[ 7].a = 5;
code[ 8].f = LIT; code[ 8].l = 0; code[ 8].a = 1;
code[ 9].f = OPR; code[ 9].l = 0; code[ 9].a = 2;
code[10].f = STO; code[10].l = 0; code[10].a = 5;
code[11].f = LOD; code[11].l = 0; code[11].a = 4;
code[12].f = LOD; code[12].l = 0; code[12].a = 3;
code[13].f = LOD; code[13].l = 0; code[13].a = 3;
code[14].f = STO; code[14].l = 0; code[14].a = 4;
code[15].f = OPR; code[15].l = 0; code[15].a = 2;
code[16].f = STO; code[16].l = 0; code[16].a = 3;
code[17].f = LOD; code[17].l = 0; code[17].a = 5;
code[18].f = LIT; code[18].l = 0; code[18].a = 3;
code[19].f = OPR; code[19].l = 0; code[19].a =10;
code[20].f = JPC; code[20].l = 0; code[20].a = 7;
code[21].f = OPR; code[21].l = 0; code[21].a = 0;
```

Fonte: Os autores.

Código 2. Instruções no vetor code para o cálculo recursivo de fibonacci de 3.

```
code[ 0].f = INT; code[ 0].l = 0; code[ 0].a =  5;
code[ 1].f = LIT; code[ 1].l = 0; code[ 1].a =  3;
code[ 2].f = STO; code[ 2].l = 0; code[ 2].a =  3;
code[ 3].f = LOD; code[ 3].l = 0; code[ 3].a =  3;
code[ 4].f = STO; code[ 4].l = 0; code[ 4].a =  8;
code[ 5].f = CAL; code[ 5].l = 0; code[ 5].a =  9;
code[ 6].f = LOD; code[ 6].l = 0; code[ 6].a =  8;
code[ 7].f = STO; code[ 7].l = 0; code[ 7].a =  4;
code[ 8].f = OPR; code[ 8].l = 0; code[ 8].a =  0;
code[ 9].f = INT; code[ 9].l = 0; code[ 9].a =  5;
code[10].f = LOD; code[10].l = 0; code[10].a =  3;
code[11].f = LIT; code[11].l = 0; code[11].a =  0;
code[12].f = OPR; code[12].l = 0; code[12].a =  9;
code[13].f = JPC; code[13].l = 0; code[13].a = 17;
code[14].f = LIT; code[14].l = 0; code[14].a =  0;
code[15].f = STO; code[15].l = 0; code[15].a =  3;
code[16].f = JMP; code[16].l = 0; code[16].a = 42;
code[17].f = LOD; code[17].l = 0; code[17].a =  3;
```

```
code[18].f = LIT; code[18].l = 0; code[18].a =  1;
code[19].f = OPR; code[19].l = 0; code[19].a =  9;
code[20].f = JPC; code[20].l = 0; code[20].a = 24;
code[21].f = LIT; code[21].l = 0; code[21].a =  1;
code[22].f = STO; code[22].l = 0; code[22].a =  3;
code[23].f = JMP; code[23].l = 0; code[23].a = 42;
code[24].f = LOD; code[24].l = 0; code[24].a =  3;
code[25].f = LIT; code[25].l = 0; code[25].a =  1;
code[26].f = OPR; code[26].l = 0; code[26].a =  3;
code[27].f = STO; code[27].l = 0; code[27].a =  8;
code[28].f = CAL; code[28].l = 0; code[28].a =  9;
code[29].f = LOD; code[29].l = 0; code[29].a =  8;
code[30].f = STO; code[30].l = 0; code[30].a =  4;
code[31].f = LOD; code[31].l = 0; code[31].a =  3;
code[32].f = LIT; code[32].l = 0; code[32].a =  2;
code[33].f = OPR; code[33].l = 0; code[33].a =  3;
code[34].f = STO; code[34].l = 0; code[34].a =  8;
code[35].f = CAL; code[35].l = 0; code[35].a =  9;
code[36].f = LOD; code[36].l = 0; code[36].a =  8;
code[37].f = STO; code[37].l = 0; code[37].a =  3;
code[38].f = LOD; code[38].l = 0; code[38].a =  3;
code[39].f = LOD; code[39].l = 0; code[39].a =  4;
code[40].f = OPR; code[40].l = 0; code[40].a =  2;
code[41].f = STO; code[41].l = 0; code[41].a =  3;
code[42].f = OPR; code[42].l = 0; code[42].a =  0;
```

Fonte: Os autores.

Código 3. Instruções no vetor code para o cálculo iterativo de fatorial de 3.

```
code[ 0].f = INT; code[ 0].l = 0; code[ 0].a = 5;
code[ 1].f = LIT; code[ 1].l = 0; code[ 1].a = 1;
code[ 2].f = STO; code[ 2].l = 0; code[ 2].a = 3;
code[ 3].f = LIT; code[ 3].l = 0; code[ 3].a = 1;
code[ 4].f = STO; code[ 4].l = 0; code[ 4].a = 4;
code[ 5].f = LOD; code[ 5].l = 0; code[ 5].a = 3;
code[ 6].f = LIT; code[ 6].l = 0; code[ 6].a = 1;
code[ 7].f = OPR; code[ 7].l = 0; code[ 7].a = 2;
code[ 8].f = STO; code[ 8].l = 0; code[ 8].a = 3;
code[ 9].f = LOD; code[ 9].l = 0; code[ 9].a = 3;
code[10].f = LOD; code[10].l = 0; code[10].a = 4;
code[11].f = OPR; code[11].l = 0; code[11].a = 4;
code[12].f = STO; code[12].l = 0; code[12].a = 4;
code[13].f = LOD; code[13].l = 0; code[13].a = 3;
code[14].f = LIT; code[14].l = 0; code[14].a = 3;
code[15].f = OPR; code[15].l = 0; code[15].a = 9;
code[16].f = JPC; code[16].l = 0; code[16].a = 5;
code[17].f = OPR; code[17].l = 0; code[17].a = 0;
```

Fonte: Os autores.

Código 4. Instruções no vetor code para o cálculo recursivo de fatorial de 2.

```
code[ 0].f = INT; code[ 0].l = 0; code[ 0].a =  5;
code[ 1].f = LIT; code[ 1].l = 0; code[ 1].a =  2;
code[ 2].f = STO; code[ 2].l = 0; code[ 2].a =  3;
code[ 3].f = LOD; code[ 3].l = 0; code[ 3].a =  3;
code[ 4].f = STO; code[ 4].l = 0; code[ 4].a =  8;
code[ 5].f = CAL; code[ 5].l = 0; code[ 5].a =  9;
code[ 6].f = LOD; code[ 6].l = 0; code[ 6].a = 10;
code[ 7].f = STO; code[ 7].l = 0; code[ 7].a =  4;
code[ 8].f = OPR; code[ 8].l = 0; code[ 8].a =  0;
code[ 9].f = INT; code[ 9].l = 0; code[ 9].a =  5;
code[10].f = LOD; code[10].l = 0; code[10].a =  3;
code[11].f = LIT; code[11].l = 0; code[11].a =  1;
code[12].f = OPR; code[12].l = 0; code[12].a =  9;
code[13].f = JPC; code[13].l = 0; code[13].a = 15;
code[14].f = JMP; code[14].l = 0; code[14].a = 24;
code[15].f = LOD; code[15].l = 0; code[15].a =  3;
code[16].f = LIT; code[16].l = 0; code[16].a =  1;
code[17].f = OPR; code[17].l = 0; code[17].a =  3;
code[18].f = STO; code[18].l = 0; code[18].a =  8;
code[19].f = CAL; code[19].l = 0; code[19].a =  9;
code[20].f = LOD; code[20].l = 0; code[20].a =  8;
code[21].f = LOD; code[21].l = 0; code[21].a =  3;
code[22].f = OPR; code[22].l = 0; code[22].a =  4;
code[23].f = STO; code[23].l = 0; code[23].a =  3;
code[24].f = OPR; code[24].l = 0; code[24].a =  0;
```

Fonte: Os autores.

REFERÊNCIAS

BROOKSHEAR, J. G. **Ciência da Computação:** uma visão abrangente. Tradução Cheng Mei Lee. 7. ed. Porto Alegre: Bookman, 2005. 515 p.

CAMPOS, A. S. **Modelo semântico de controle de fluxo e chamada de sub-rotina para Hardware virtual**. Trabalho de Conclusão de Curso (Graduação) - Ciências da Computação, Universidade Estadual de Santa Cruz, Ilhéus,BA, 2022a.

CAMPOS, A. S. **AllanaVM**. 2022b. Disponível em: https://github.com/AllanaCampos/VMs. Acesso em: 06 out. 2023.

CAPUANO, F. G.; IDOETA, I. V. **Elementos de eletrônica digital**. 41. ed. São Paulo: Érica, c2012. 524 p.

HIALINX. **O surgimento da virtualização e as grandes mudanças que ela teve**. 2019. Disponível em: https://www.hialinx.com.br/post/o-surgimento-da-virtualização-e-as-grandes-mudanças-que-ela-trouxe. Acesso em: 07 out. 2022.

IBM. **System/360 From Computers to Computer Systems**. 2023. Disponível em: https://www.ibm.com/ibm/history/ibm100/us/en/icons/system360/. Acesso em: 07 out. 2023.

LOGIC.LY. **A logic circuit simulator for Windows and macOS - logic gates, flip-flops, computer architecture, electronics, integrated circuits**. 2021. Disponível em: https://logic.ly/. Acesso em: 06 out. 2023.

MICROSOFT. **Azure**. 2023. Disponível em: https://azure.microsoft.com/pt-br/resources/cloud-computing-dictionary/what-is-azure/. Acesso em: 07 out. 2023.

Nand2Tetris. **Building a Modern Computer From First Principles**. 2021. Disponível em: https://www.nand2tetris.org/. Acesso em: 06 out. 2023.

NELSON, V. *et al*. **Digital Logic Circuit Analysis and Design**. [S.l.]: Prentice Hall, 1995.

ORACLE. **Oracle VM**. 2021. Disponível em: https://docs.oracle.com/en/virtualization/oracle-vm/3.4/concepts/E64081.pdf. Acesso em: 07 out. 2023.

SANTANA, I. S. S. **Programa p-code que calcula o fatorial iterativo de 5**. [*S. l.: s. n.*], 2020. Disponível em: http://cpp.sh/8xz7z. Acesso em: 7 out. 2023.

SANTANA, I. S. S. **Vídeo no site youtube sobre a simulação da ULA no software Logic.ly**. [*S. l.: s. n.*], 2021a Disponível em: https://youtu.be/xXietuJC83o. Acesso em:

SANTANA, I. S. S. **Vídeo no site youtube sobre o teste automatizado da ULA de 4 bits.** Disponível em: https://youtu.be/KRKOzmZ2_fc. 2021b.

SANTANA, I. S. S. **Programa p-code que calcula a Sequência de Fibonacci iterativo**. [*S. l.*: *s. n.*] 2021c. Disponível em: http://cpp.sh/4lucc.

SEBESTA, R. W. **Conceitos de linguagens de programação**. São Paulo: Bookman, 2018.

SINGH, H. **Next-Gen Virtualization for Dummies, Vmware Special Edition**. New Jersey: John Wiley & Sons, Inc. 2019. 64 p.

VMWARE. **VMware**. 2023. Disponível em: https://www.vmware.com/. Acesso em: 07 out. 2023.

WIKIPEDIA. **Máquina Virtual Java**. 2023. Disponível em: https://pt.wikipedia.org/wiki/M%C3%A1quina_virtual_Java. Acesso em: 08 out. 2023.

WIRTH, N. **Compiler Construction**. Addison-Wesley, 1996.

ZANOELLO, T. B. **Curso de introdução a criação e uso de Máquinas Virtuais**. 1. ed. São Paulo: Seção Técnica de Informática – USP, 2017. 28 p.