Desenvolvimento de uma ferramenta de simulação de processadores para uso didático em cursos de computação

Morgana Sartor* Marcelo Daniel Berejuck†

2021, setembro

Resumo

O processo de aprendizagem de Organização e Arquitetura de Computação é fundamental para estudantes de Engenharia de Computação e áreas relacionadas. A complexidade e a falta de familiaridade dos estudantes com o conteúdo tornam difícil a compreensão de conceitos essenciais para o desenvolvimento de habilidades fundamentais ao profissional que atua nesta área da Engenharia, e que hoje tem que lidar com novos paradigmas computacionais, como a Internet das Coisas. Neste contexto, esse trabalho apresenta uma ferramenta didática de simulação de processadores para ensino na disciplina de Organização e Arquitetura de Computadores, com ênfase em processadores para em sistemas embarcados. São simulados dois processadores. Um didático, desenvolvido para facilitar o entendimento de um processador, e outro baseado em um modelo comercial desenvolvido pela empresa Britânica *ARM Holdings*. Através de pesquisa bibliográfica foi feita uma comparação entre processadores didáticos publicados na literatura, para nortear o desenvolvimento da ferramenta. São apresentados ainda os resultados preliminares da aplicação do processador didático proposto neste trabalho, que foi usado em sala de aula no curso de Organização e Arquitetura de Computadores do curso de Engenharia de Computação da UFSC.

Palavras-chaves: Arquitetura e Organização de Computadores: Ferramenta Didática; Simulador de Processador.

*morgana.sartor@outlook.com
†marcelo.berejuck@ufsc.br

Development of a processor simulation tool for didactic use in computing courses

Morgana Sartor* Marcelo Daniel Berejuck†

2021, setembro

Abstract

The learning process of Computer Organization and Architecture is fundamental for students of Computer Engineering and related areas. The complexity of the theme and lack of familiarity of the students with the content make it difficult to understand essential concepts important to develop fundamental skills for a computing area professional dealing with new computational paradigms, such as the Internet of Things. In this context, this work presents a didactic tool to simulate processors for teaching the subject of Computer Architecture and Organization, emphasising processors for embedded systems. Two processors are simulated. One is didactic, developed to facilitate the understanding of a processor, and the other is based on a commercial model developed by the British company ARM Holdings. Through bibliographic research, a comparison between didactic processors published in the literature was made to guide the development of the tool. The preliminary results of practical use of the tool proposed in this work, which was used in a classroom in the Computer Architecture and Organization course of the Computer Engineering course at UFSC, are also presented.

Key-words: Computer Architecture and Organization; Didactic Tool; Processor Simulator.

1 Introdução

A Internet das Coisas (IoT) é um paradigma que cobre quase todas as áreas da vida humana nos dias de hoje, da educação à saúde, dos serviços aos produtos, de casas às cidades inteligentes atingindo qualquer área na qual a IoT seja aplicável, como automotiva, infra-estrutura e negócios. Enfim, os dispositivos de IoT estão cada vez mais pervasivos (CHéOUR et al., 2020). O desenvolvimento de dispositivos de IoT demandam

*morgana.sartor@outlook.com
†marcelo.berejuck@ufsc.br

conhecimentos específicos na área computacional, como o projeto de sistemas de tempo real, o uso de microcontroladores, o uso de dispositivos lógicos programáveis, entre outras tecnologias. Assim, é fundamental que os profissionais formados para atuar nesta área possuam um bom entendimento de como técnicas e tecnologias para sistemas embarcados podem ser utilizadas na concepção de tais dispositivos.

Porém, o processo de aprendizado dessas técnicas e tecnologias não é algo trivial, uma vez que o funcionamento interno de computadores embarcados pode variar em termos de recursos computacionais ou conectividade. Neste sentido, ferramentas que oferecem abordagens no ensino de um conteúdo de difícil abstração se apresentam como opções interessantes para a docência, contanto que ela torne o ensino do conteúdo abordado mais prático e abrangente, despertando o interesse do aluno e aumente sua vontade por buscar informação (SANTOS; COSTA, 2006). Este é um desafio da atualidade que se apresenta ao ensino de Organização e Arquitetura de Computadores (OAC), uma disciplina curricular obrigatória presente na grade dos cursos de graduação em Engenharia Elétrica, Eletrônica e de Computação: ensinar conceitos fundamentais na concepção de sistemas embarcados, mantendo os estudantes motivados para evoluírem no aprendizado destes conceitos.

A disciplina de Organização e Arquitetura de Computadores tradicionalmente apresenta os conceitos básicos sobre o funcionamento de processadores, fazendo uso de arquiteturas como o tradicional 8086 ou o processador MIPS. As ferramentas de montagem e simulação geralmente permitem que os estudantes manipulem dados de forma lógica e matemática, entre registradores e posições virtuais de memória de dados. No entanto, sistemas embarcados são projetados como sistemas de tempo real. Isso significa que o mecanismo de interrupção de hardware é uma técnica crucial para o atendimento dos requisitos temporais destes sistemas, que muitas vezes não possuem sistema operacional em execução para lidar com tarefas de tempo real, como é o caso dos microcontroladores.

Num levantamento bibliográfico sobre ferramentas de montagem e simuladores, apresentado na Seção 2, observou-se que a maioria deles tem uma arquitetura simplificada em termos de linguagem de máquina e uma organização limitada em termos de interfaces externas, tais como pinos de entrada e saída e outras interfaces. Além disso, tais simuladores não evidenciam como os processadores simulados fazem o tratamento de interrupções de hardware, e a maioria deles não permite que os usuários façam mudanças na sua organização, nem os integre com outros sistemas, por exemplo, um microprocessador físico ou sintetizável.

Neste contexto, este documento apresenta os resultados parciais obtidos no desenvolvimento de uma ferramenta de simulação de processadores para uso didático em cursos de graduação em Engenharia Elétrica, Eletrônica e de Computação. Esta ferramenta permite a simulação de duas arquiteturas: uma didática, chamada de μPD, e outra baseada em um processador comercial utilizado no desenvolvimento de sistemas embarcados, chamada de Cortex M0®, do fabricante ARM®, cujo desenvolvimento foi executado em duas partes.

A primeira parte contempla o processador da arquitetura didática μPD, com a implementação de um ambiente de simulação visual e uma integração com geração de código VHDL (acrônimo de Linguagem de descrição de hardware VHSIC *Very High Speed Integrated Circuits*) de seus componentes internos. O conjunto de instruções do μPD inclui operações lógicas e matemáticas, busca e armazenamento de dados em memória operações de desvio, chamadas de funções e interrupções de hardware. A ferramenta de simulação possibilita ao usuário visualizar o funcionamento interno do processador durante a execução de um programa escrito em linguagem de montagem Assembly, instrução por instrução. A segunda parte contempla o desenvolvimento de ambientes, similares ao μPD, para a

arquitetura Cortex M0. A proposta é ter um simulador que mantenha-se fiel à linguagem Assembly original do Cortex M0, apenas modificando as instruções binárias geradas pelo compilador, permitindo um *tour* pelo processador comercial na execução de suas instruções.

Para uma melhor organização e apresentação, o documento foi dividido em seções. A seção 2 apresenta os principais trabalhos relacionados ao uso de simuladores didáticos encontrados na literatura. São apresentados os materiais e métodos utilizados neste trabalho na seção 3. A seção 4 apresenta a primeira parte do desenvolvimento, composta apenas do processador didático μPD e seu ambiente de simulação e geração de código de hardware sintetizável. A segunda parte do desenvolvimento é feita na seção 4.2, com a implementação do processador Cortex M0, onde também foram destacados os pontos positivos e apontados possíveis melhorias. Os resultados obtidos em uma primeira avaliação da ferramenta com o processador μPD com estudantes de graduação em Engenharia de Computação na disciplina de Organização e Arquitetura de Computadores são apesentados na seção (5), em formato de gráficos e tabelas. Na última seção (6) fazem-se presentes as conclusões e trabalhos futuros.

2 Trabalhos Relacionados

Nesta seção serão apresentados os trabalhos relacionados, descrevendo os principais pontos de cada arquitetura e ambiente de simulação proposto pelos autores. Na tabela 1 será apresentado um comparativo entre os trabalhos relacionados citados e a ferramenta que está sendo proposta neste trabalho.

2.1 Mars

O simulador Mars (VOLLMAR; SANDERSON, 2006) apresenta um ambiente de desenvolvimento integrado com uma plataforma de simulação da arquitetura do processador MIPS (HENNESSY; PATTERSON, 2014) com conjunto de instruções reduzido. As instruções presentes são somente as educacionalmente importantes, não sendo implementadas pseudo-instruções ou consideradas de uso mais profissionais. Entre as funcionalidades de simulação do Mars estão: execução passo a passo do código do usuário com velocidade variável, visualização das alterações dos trinta e dois registradores e da memória durante a execução, mudança da visualização dos dados de decimal para hexadecimal, menu de navegação na memória.

2.2 CompSim

O CompSim (ESMERALDO; LISBOA, 2017) é um ambiente de simulação para apoio ao ensino de OAC, baseado em uma arquitetura pré-definida de hardware simulável e parametrizável chamada Mandacaru. A arquitetura do processador, que foi desenvolvida pelos autores, é composta de processador, memória RAM, memória cache, periféricos de entrada e saída e uma estrutura de comunicação composta de dois barramentos.

O processador da Mandacaru é chamado de Cariri e é baseado em acumulador, tendo 16 instruções monociclo de baixo nível, operandos inteiros de 16 bits com sinalização, espaço de endereçamento diferenciado para entrada e saída e memória principal e suporte aos modos de endereçamento imediato, direto, indireto, via registrador e implícito. Ele possui palavra de dados de 16 bits, uma ULA, uma unidade de controle, um somador, um banco de registradores, um acumulador, um PC, alguns registradores de instrução,

endereçamento e *buffer* de memória, endereçamento e *buffer* de IO, apontador de pilha, *flag* de status de operação e segmento de código, dados e pilha.

A interface gráfica do CompSim é composta de um editor, uma tabela que representa os registradores, uma tabela que representa a memoria RAM, uma tabela que representa a memoria cache, componentes gráficos para interações com o programa, controles para configuração e gerenciamento da simulação, exibição de estatísticas e eventos da simulação, exibição da pilha, exibição das variáveis e um *Assembly Report*, que mostra um relatório gerado a partir do programa do usuário.

2.3 Bipide

Possibilitando a simulação de códigos em linguagem Portugol sobre a arquitetura BIP (PEREIRA, 2008), o Bipide (PERES; ZEFERINO; VIEIRA, 2017) disponibiliza um ambiente de desenvolvimento integrado acoplado à plataforma de simulação. Sua estrutura é composta por: um editor de textos, um compilador que traduz a linguagem Portugol para a linguagem do BIP, um simulador dos processadores BIP capaz de simular as instruções desses e exibir graficamente por meio de animações e um módulo de ajuda.

O Bipide apresenta em seu ambiente três abas, sendo uma para cada um dos três módulos: programação, simulação e ajuda. O módulo de programação oferece ao usuário recursos e funcionalidade típicas de editores de código, opções de gerenciamento e edição de arquivos e relatório de compilação. O ambiente de simulação possibilita a execução passo a passo, oferecendo a visualização do código em alto nível e em Assembly. O módulo de ajuda é composto por informações teóricas a respeito da arquitetura e organização de computadores e apresenta a descrição dos processadores BIP.

2.4 FRISCjs

Baseado no processador FRISC (BASCH DANKO ; KOVAč, 2004) de arquitetura RISC, a aplicação FRISCjs (ZUZAK, 2016) possui duas partes: um assembler para o FRISC que converte o código assembly para o código de máquina, e o simulador do processador FRISC que executa o código de máquina. A interface do simulador é composta por quatro páginas: a interface de simulação, uma interface de *load* e *save* de arquivos na sua máquina local, uma seção de documentação para quem está utilizando a ferramenta, uma aba de exemplos de código e uma última que explana sobre o projeto.

A página de simulação da aplicação possui um editor, que grifa as palavras reservadas da linguagem assembly do processador. Além disso, possui um menu de simulação, com botões de *load* − que é usado para carregar o programa presente no editor −, executar − tudo ou um por um −, pause, *stop*, limpar logs e configurações. Durante a execução das instruções é possível ver o conteúdo dos registradores de acordo com o fluxo da instrução que está sendo executada, além de ser possível interagir com o programa usando os campos de entrada e saída.

2.5 SimuS

A ferramenta proposta por (BORGES; SILVA, 2016) apresenta um processador, desenvolvido tomando como base o Neander-X (BORGES; SILVA, 2006), chamado Sapiens, cuja arquitetura é baseada em acumulador. Esse processador tem tamanho de instrução variável (de 1 a 3 bytes), quatro tipos de endereçamento e 31 instruções − sendo mantidas todas as instruções do Neander-X. A ferramenta apresentada pelo autor é muito semelhante

ao trazido por (BORGES; SILVA, 2006), mantendo-se os mesmos módulos. e aparência simples, apenas com acréscimo de módulo de entrada e saída. Dentre os módulos mantidos, tem-se: editor de texto integrado, montador com compatibilidade para códigos escritos para o processador, simulador da arquitetura habilitado para visualização e modificação dos elementos estruturais do processador e da memória simulada, módulo de depuração, visualização e modificação da memória simulada, ferramenta de apoio ao aprendizado de instruções, simulador de dispositivos de I/O, e gerador/carregador de imagem da memória simulada.

2.6 CESAR e RAMSES

O computador CESAR, existente em duas versões − CESAR16 e CESAR16i − é um processador didático desenvolvido por (WEBER, 2001) com o intuito de facilitar o aprendizado de OAC. Algumas características são comuns em ambas versões do processador, incluindo o tamanho da palavra de dados de 16 bits, a representação dos dados em complemento de dois, os 8 modos de endereçamento nativo adicionando aos 4 modos derivados, a memória RAM com 65535 endereços, assim como a ROM, os dois periféricos de I/O, e o conjunto de instruções − 41 no total. A versão CESAR16i incorpora um conjunto de registradores na memória, necessários para gerenciar o mecanismo de interrupção. O ambiente gráfico do simulador de CESAR é simplista e composto por: representação dos registradores, ROM, RAM, periféricos de I/O, edição e carregamento de código em memória, e gerenciamento da dinâmica das simulações.

O processador didático RAMSES (WEBER, 2001) possui o mesmo intuito que o já citado CESAR e possui algumas características em comum, como representação dos dados em complemento de dois. Em contrapartida, possui somente 8 bits de palavra de dados, quatro modos de endereçamento − direto, indireto, imediato e indexado −, e um conjunto de instruções de 16 instruções. O RAMSES não possui mecanismos de interrupção e sua memória de programa é pequena (255 posições), bem como sua memória de programa. Seu simulador possui ambiente gráfico semelhante ao do CESASR, com diferença de apresentar um pequeno guia de suas instruções e nenhuma representação de entrada e saída, o que já era esperado visto que esse processador não apresenta periféricos.

2.7 COCONUT

Os autores (RADIVOJEVIC; STANISAVLJEVIC; PUNT, 2018) propuseram um simulador educacional configurável para organização e arquitetura de computadores, chamado COCONUT. Segundo eles, os estudantes podem criar seus próprios processadores com uma arquitetura arbitrária e organização simples no nível de transferência de registradores, escrever um programa em assembly para ser executado no processador e observar nele as fases de execução de instrução do programa. Os estudantes criam um processador em VHDL usando alguns blocos básicos e seguindo algumas regras de conexão. Um simulador está disponível para avaliar e testar o processador implementado. O fato de ser configurável permite que os alunos usem o simulador para definir a decodificação da instrução do processador da unidade de controle e o conteúdo da memória de programa do processador.

2.8 ARMSim#

ARMSim#, apresentado por (HORSPOOL; LYONS; SERRA, 2009), foi desenvolvido como uma aplicação *desktop* para rodar em um ambiente Windows. Ele permite que os usuários simulem a execução de programas desenvolvidos na linguagem assembly do

ARM em um sistema baseado no processador ARM7TDMI, em conjunto com a coleta de estatísticas para execução e uso de cache. Isso também pode ser feito através de *plugins*, que fornecem um ambiente interativo de I/O simulando exemplos de sistemas embarcados. Os autores implementaram vários *plugins*, que estão disponíveis para *download*. Os *plugins* que adicionam um módulo gráfico, tais como o que permite a execução de interfaces interativas e o que suporta o processo de interrupções, são funcionalidades importantes. A visão principal é composta de um painel contendo um processador ARM e um conjunto de periféricos − botões, luzes de LED, teclado, pequenas telas LCD.

2.9 EduMIPS64

Os autores (PATTI et al., 2012) propuseram o simulador visual EduMIPS64 como suporte aos estudantes de graduação do curso de arquitetura de computadores. Ele implementa um pipeline de cinco estágios, permitindo o paralelismo em nível de instrução. Como cada ciclo virtual de CPU é executado,a interface de usuário atualiza todos seus componentes para mostrar a representação do status atual da execução. O simulador foi desenvolvido na linguagem Java e é uma implementação de interface gráfica que fornece, além dos registradores e memória necessários, algumas estatísticas de execução como o número de ciclos por instrução.

2.10 MarieSim

Os autores (NULL; LOBUR, 2003) desenvolveram o MarieSim: um simulador de arquitetura baseado na arquitetura MARIE e projetado para ensinar o básico organização e arquitetura de computadores. Ele é um ambiente gráfico que permite os estudantes visualizarem como as instruções em linguagem assembly afetam os registradores e a memória de um sistema computacional. O ambiente gráfico do MarieSim e o simulador de caminho de dados que o acompanha foram escritos em Java Swing, e seu *assembler* integrado foi escrito em Java. MARIE é um acrônimo para "*Machine Architecture Really Intuitive and Eaasy*".

2.11 Visão Geral dos Trabalhos Relacionados

Esta subseção resume os simuladores acadêmicos utilizados como trabalhos relacionados na Tabela 1, delineando as principais características de cada simulador. É importante destacar aqui que somente as ferramentas de simulação Mars, EduMIPS64 e ARMSim são baseadas em arquiteturas comerciais. Por outro lado, as demais possuem arquiteturas proprietárias desenvolvidas estritamente para uso didático, muitas delas não possuindo os recursos necessários para o ensino de sistemas de tempo real e sistemas embarcados. Como pode ser visto na tabela, não há ferramenta que apresente interface de interrupção em conjunto com integração com VHDL.

Na seção de conclusão e trabalhos futuros, na tabela 8, a tabela 1 é acrescida da ferramenta desenvolvida na primeira etapa deste projeto, bem como de sua evolução. Também são feitos alguns comentários sobre as principais diferenças.

3 Materiais e Métodos

Nas últimas três décadas, a *Association for Computing Machinery* (ACM) e a *Computer Society of the Institute for Electrical and Electronics Engineers* (IEEE-CS) uniram

Tabela 1 – Comparativo entre as ferramentas de simulação publicadas.

	Mars	CompSim	Bipide	FRISCjs	SimuS	CESAR	RAMSES	COCONUT	ARMSim#	EduMIPS64	MarieSim
Editor de código	✓	✓	✓	✓	✓	-	-	✓	✓	-	✓
Compilador/Tradutor/Montador	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
Execução passo-a-passo	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
Simulação em blocos	-	-	✓	-	-	-	-	-	✓	-	-
Integração com VHDL	-	-	-	-	-	-	-	✓	-	-	-
Interface I/O	-	-	-	✓	✓	✓	-	-	-	-	-
Interface de Interrupção	-	-	-	✓	-	✓	-	-	-	-	-
Visão do caminho de dados	-	-	✓	✓	-	-	-	-	-	-	✓
Baseado em acumulador	-	✓	✓	-	✓	-	-	-	-	-	✓
Baseado em registrador	✓	-	-	✓	-	✓	✓	✓	✓	✓	-
Multi Arquitetura	-	-	-	-	-	-	-	-	-	-	-

esforços para promover e atualizar periodicamente um documento com diretrizes curriculares chamado "Currículo de Computação"(PRASAD P. W. C.; CHAN, 2016),(IMPAGLIAZZO R. SLOAN; SRIMANI, 2003). Estas diretrizes estão definindo amplas áreas de conhecimento que se aplicam a todos os programas de engenharia de computação. De acordo com (PRASAD P. W. C.; CHAN, 2016), as seis unidades de conhecimento que precisam ser trabalhados pelos cursos de engenharia de computação são: Fundamentos da arquitetura de computadores, Organização e arquitetura do sistema de memória, Interface e comunicação, Subsistemas de dispositivos, Projeto de sistemas com processadores e Organização da CPU.

O curso de graduação em Engenharia de Computação da UFSC possui em sua grade curricular três disciplina que cobrem, em diferentes níveis de complexidade, estas unidades de conhecimento: Organização e Arquitetura de Computadores (OAC), Linguagem de Síntese de Hardware (LSH), e Projeto de Sistemas Embarcados (PSE). Apesar destas três disciplinas demandarem muitos outros conhecimentos distintos, ainda assim elas mas podem utilizar da mesma ferramenta de simulação em suas ementas. Por exemplo, além de OAC, LSH poderia explorar o conceito de *softcore* e o desenvolvimento de componentes em VHDL para conectar aos processadores, enquanto que PSE poderia utilizar o desenvolvimento de um sistema embarcado com *softcore* em FPGA. Assim, entendemos que a ferramenta de simulação proposta deve apresentar funcionalidades que auxiliem no aprendizado destas unidades de conhecimento e possam ser utilizada como suporte didático no ensino de OAC, LSH e PSE. Para atingir este objetivo, entende-se que são recursos necessários na ferramenta para ambos os processadores (μPD e Cortex M0):

- Um editor para permitir a edição de programas em linguagem de montagem (Assembly);
- Uma tela de simulação mostrando as áreas de código, de dados e registradores de cada processador;
- Um gerador de código VHDL contendo um componente com o código de máquina para cada processador; e
- Uma tela que permita ao usuário acompanhar o funcionamento do processador através de blocos funcionais.

Com o objetivo de prover uma ferramenta de simulação que traga uma arquitetura didática compatível com uma arquitetura mais complexa, a estratégia adotada foi dividir o projeto em duas fases. Uma delas totalmente voltada a uma arquitetura didática que possuísse elementos de um processador de alto nível, sendo desenvolvido o processador em linguagem de síntese de hardware, uma ferramenta de simulação para o mesmo e por fim levando essa arquitetura e sua ferramenta para a sala de aula. A segunda etapa trata da evolução da ferramenta de simulação, bem como a introdução da arquitetura comercial em conjunto à arquitetura didática já desenvolvida.

4 Ferramenta de Simulação proposta

4.1 Simulador para o processador μPD

O processador didático μPD foi idealizado para ser simples. Ele é composto por uma arquitetura de hardware, desenvolvida especificamente para uso didático, e um ambiente

gráfico que disponibiliza uma visão interna dos componentes do processador na execução de suas instruções. As próximas subseções detalham a organização interna da ferramenta, arquitetura e organização do processador e as funcionalidades oferecidas no software de simulação.

4.1.1 Visão geral do processador

O μPD é um processador monociclo de arquitetura RISC (*Reduced Instruction Set Computer*), com acesso à uma unidade de memória ROM, um barramento de entrada e saída para periféricos. A memória de dados é interna ao processador com $1KByte$ de capacidade de armazenamento. Buscou-se aqui desenvolver um processador que tivesse todos os recursos que um processador de alto nível teria de forma que preservasse sua simplicidade e forma didática. Dessa forma, temos um processador com quantidade de registradores reduzida, contendo seis interrupções e barramentos de entrada e saída.

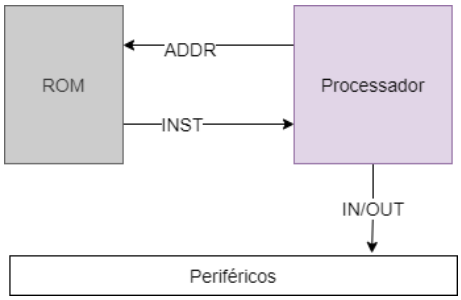


Figura 1 – Organização proposta para o processador μPD.

A ROM (*Read Only Memmory*) permite armazenar até $0.5KWords$ ou $1Kbyte$ e armazena 16 bits por endereço. Ela possui 512 endereços disponíveis para armazenamento de programa, desde a rotina principal até as funções, e também conta com mais 6 áreas reservadas às interrupções, como é mostrado na Tabela 2.

Tabela 2 – Endereçamento da memória ROM

Área	Endereço inicial (hex)	Endereço final (hex)
Programa	000	1FF
INT(0)	200	2FF
INT(1)	300	3FF
INT(2)	400	4FF
INT(3)	500	5FF
INT(4)	600	6FF
INT(5)	700	7FF

4.1.2 Arquitetura e Organização do Processador

O processador μPD possui arquitetura simples mas que contempla todos os tipos de instruções de processamento e controle, sendo adequado para ensino de programação em linguagem de máquina, com possibilidade de implementar rotinas de diversos graus de complexidade. A organização interna do processador é composta de dois caminhos lógicos denominados de “caminho de controle” e “caminho de dados”, ambos representados na Figura 2. Dentre as características do processador didático μPD, estão:

- Arquitetura RISC;
- Palavra de dados de 16 bits;
- 21 instruções monociclo de baixo nível (*Instruction Set Architecture*);
- Operandos inteiros de 9 bits com sinalização;
- Suporte aos modos de endereçamento absoluto e via registrador; e
- Permissão de uso de até 6 interrupções;

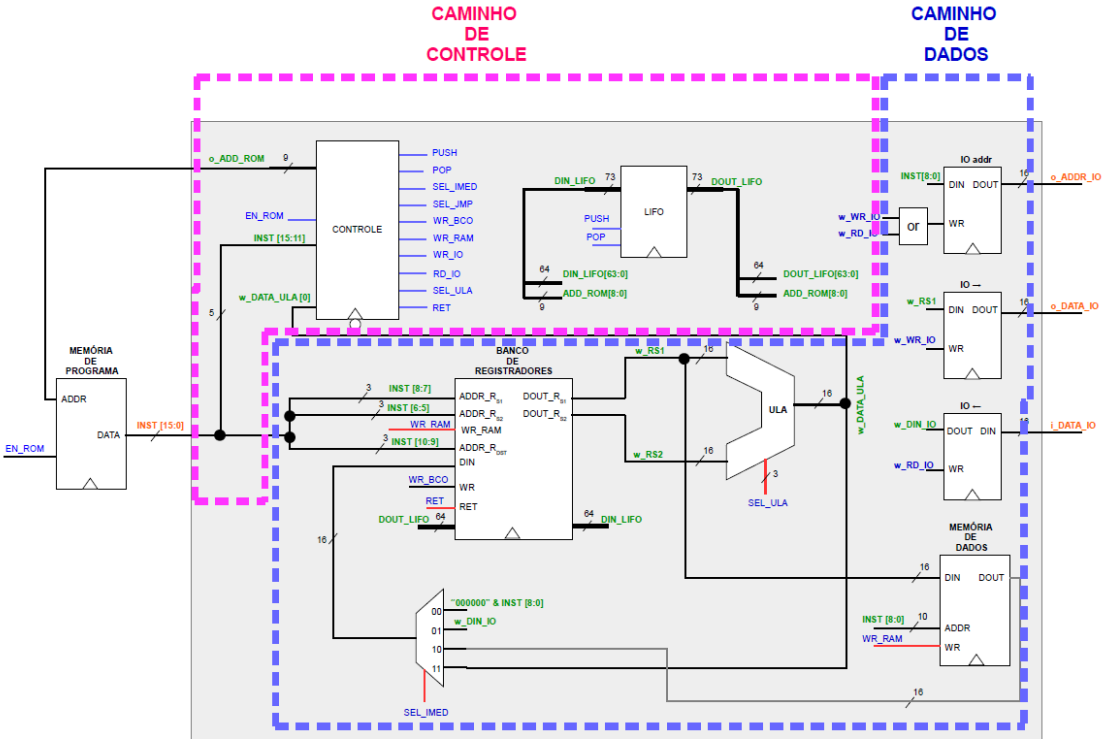


Figura 2 – Processador uPD

O caminho de controle é responsável por buscar a instrução na ROM, que é externa ao processador, e controlar o fluxo de execução das instruções no caminho de dados. Seus principais componentes internos são a unidade de controle e uma memória do tipo LIFO (*Last in, First Out*), usada para salvar o contexto do sistema quando é feita uma chamada de sub rotina ou de uma interrupção.

Como já citado, os dados são processados em si dentro do caminho de dados, que é comandado por sinais de controle originados no caminho de controle. Ele é composto de um banco de registradores, possuindo quatro registradores, uma ULA (Unidade Lógica e Aritmética) e registradores que fazem interface com pinos de entrada e saída, possibilitando a conexão com até quinhentos e doze dispositivos nessa porta.

Em suas instruções, os cinco bits mais significativos indicam o OPCODE (Código de Operação) e os demais bits codificam os operandos. Na Figura 3 são mostrados os formatos de instrução existentes no processador e na tabela 3 encontra-se o detalhamento de todas elas, com seus respectivos códigos e descrições.

Tabela 3 – ISA do processador μPD

Mnemônico	OPCODE	Descrição
NOP	00000	"No operation". Processador gasta um ciclo de relógio se fazer nenhuma operação.
LDI	00001	Carrega valor imediato contido na instrução (bits de 0 a 8) em um determinado Registrador (RDST)
ADD	00010	Soma o conteúdo de dois registradores e armazena o resultado em um terceiro registrador.
SUB	00010	Subtrai o conteúdo de dois registradores e armazena o resultado em um terceiro registrador.
OUT	00100	Escreve o valor contido em um registrador (R0, R1, R2 ou R3) nos pinos de I/O.
IN	00101	Lê o valor contido na porta de I/O em um dos registradores (R0, R1, R2 ou R3).
JI	00110	Salto (jump) incondicional. Salta para o endereço contido na instrução (bits de 0 a 8).
LD	00111	Carrega conteúdo armazenado na memória RAM em um dos registradores (R0, R1, R2 ou R3). O endereço da memória RAM está contido na instrução (bits de 0 a 8).
STO	01000	Armazena conteúdo de um registrador (R0, R1, R2 ou R3) em uma posição de memória RAM. O endereço da memória está contido na instrução (bits de 0 a 8).
JZ	01001	Salto condicional (jump) se o conteúdo de um registrador (R0, R1, R2 ou R3) for igual a 0 (zero). O endereço do salto está na instrução (bits de 0 a 8).
JE	01010	Salto condicional (jump) se o conteúdo de um registrador (R0, R1, R2 ou R3) for igual a 1 (um). O endereço do salto está na instrução (bits de 0 a 8). É necessária a execução da instrução CMP antes da instrução JE.
AND	01011	Executa E lógico entre o conteúdo de dois registradores, armazenando o resultado em um terceiro registrador.
OR	01100	Executa OU lógico entre o conteúdo de dois registradores, armazenando o resultado em um terceiro registrador.
XOR	01101	Executa OU EXCLUSIVO lógico entre o conteúdo de dois registradores, armazenando o resultado em um terceiro registrador.
NOT	01110	Executa NÃO lógico no conteúdo de um registrador, armazenando o resultado em um segundo registrador.
CALL	01111	Chamada de sub rotina (função). Endereço de salto está na instrução (bits de 0 a 8).
RETI	10000	Retorno de interrupção. Volta para o ponto onde estava sendo executado o programa antes da ocorrência da interrupção.
SETR	10001	Configura registradores internos do processador. No momento existe apenas o INT, que habilita interrupções externas.
STOP	10010	Para a execução do programa por tempo indeterminado.
RET	10011	Retorno de sub rotina (função). Retorna para o endereço de onde foi chamada a última sub rotina (onde foi executado o CALL).
CMP	10100	Compara o conteúdo armazenado em dois registradores quaisquer (R0, R1, R2 ou R3). Este comando deve obrigatoriamente preceder o comando JE.

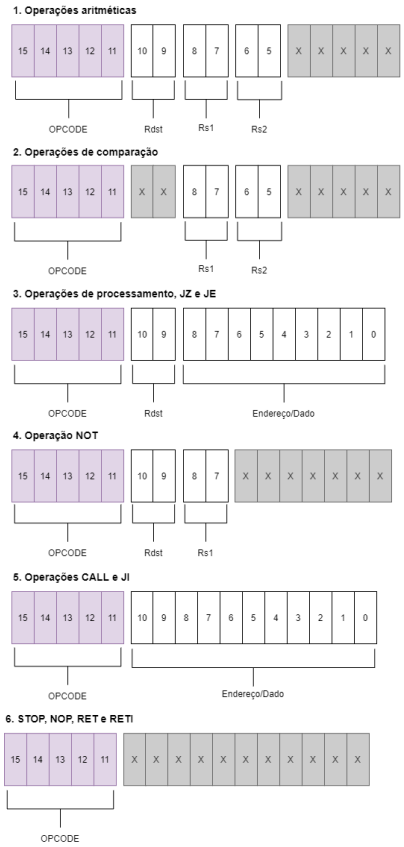


Figura 3 – Formatos de instrução no processador μPD

É importante ressaltar que faz-se presente instruções de configuração para habilitar interrupções (SETR), bem como instruções de entrada e retorno de interrupção − IN e RETI respectivamente. Elas permitem que o processador seja facilmente integrado ao ensino de sistemas embarcados, aplicando ao conteúdo de interrupções e as permissões necessárias para que as mesmas sejam usadas via software, e como elas podem ser chamadas. Além dessas, a ISA também inclui operações aritméticas, lógicas, de carregamento de dados, de saltos no fluxo de execução e de entrada e saída. Com todas as instruções presentes é possível implementar desde os programas mais simples até os muito complexos.

4.1.3 O Software de Simulação para o μPD

A interface da primeira versão que contempla apenas o μPD foi desenvolvida utilizando os formulários do Windows®, podendo ser executado em qualquer sistema operacional por meio de emuladores de programas executáveis. Os principais componentes gráficos utilizados no simulador são:

- Editor de código: utilizado para edição e criação, bem como carregamento externo de código de baixo nível Assembly. Possui todas as funcionalidades de editores de código padrão;
- Execução por texto: após ser gerado o código binário, a tabela com informações referentes à memória de programa é carregada bem como a tabela do banco de

registradores e da memória RAM, além do valor do registrador especial PC. É possível converter os valores das tabelas de decimal para hexadecimal.;

- Execução de blocos: a execução por blocos exibe o esquemático do diagrama de blocos do processador e, sempre que uma instrução é executada, os pinos utilizados nos componentes em seu caminho de dados são destacadas em verde e algumas possuem seu valor explícito; e
- Mensagens: exibe informações competentes ao código ou à execução do mesmo.

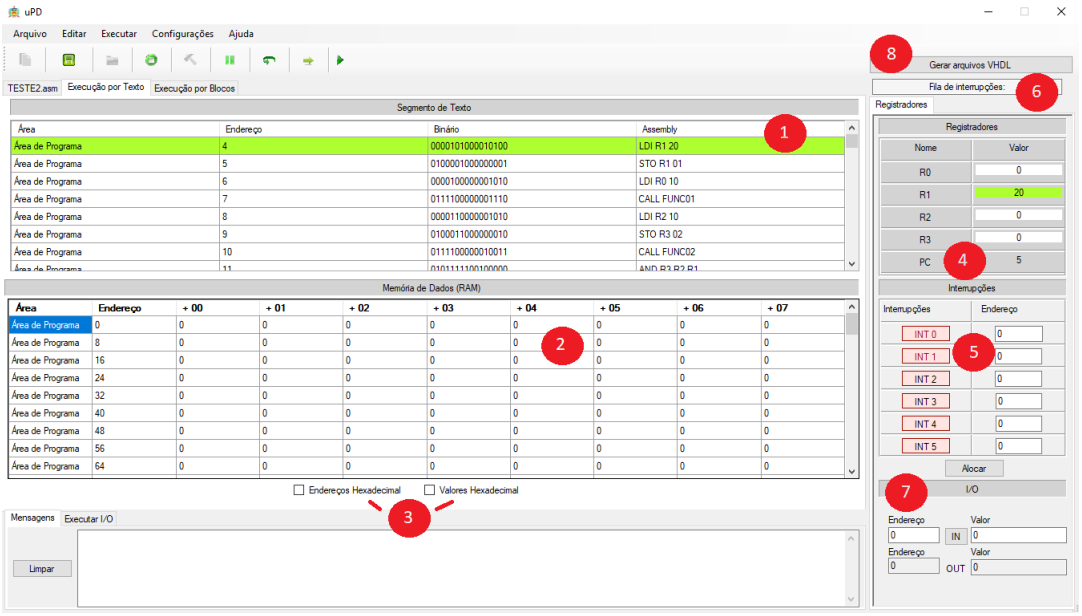


Figura 4 – Exemplo de execução por texto.

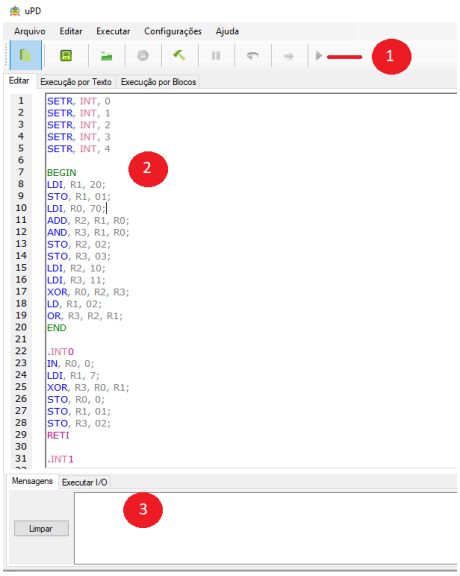


Figura 5 – Exemplo de execução por texto.

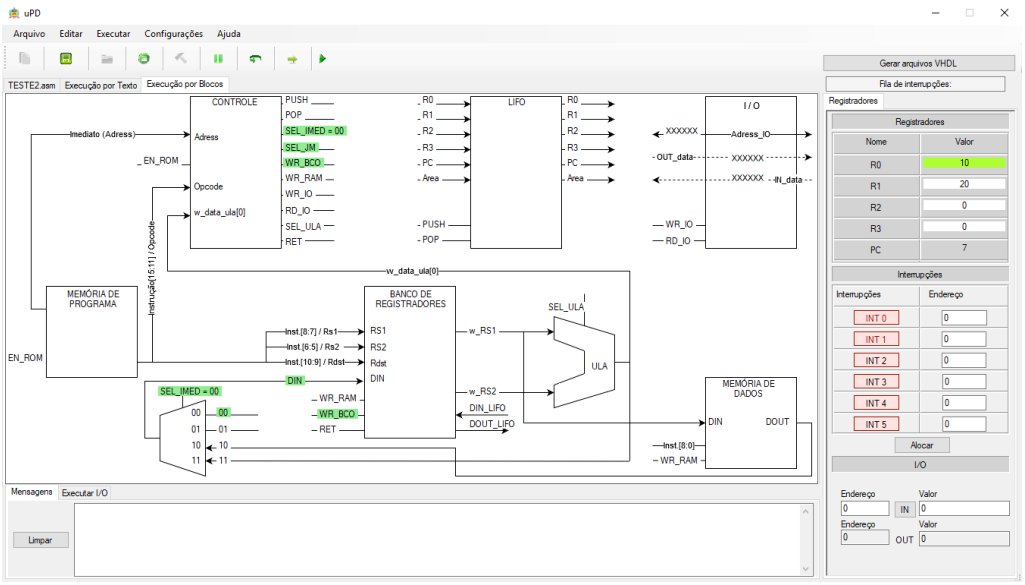


Figura 6 – Exemplo de execução por blocos.

Na Figura 4 há um *snapshot* do software de simulação do μPD durante a execução de um programa contendo interrupções, onde pode-se observar os destaques da instrução que está sendo executada e do registrador que está sendo alterado. Nessa imagem são apresentados os principais componentes do ambiente de simulação por texto do μPD: 1) memória de programa detalhada, 2) memória RAM (*Random Access Memmory*), 3) flag para mudança no estilo da apresentação dos números, 4) banco de registradores detalhado, 5) interface de interrupções, 6) fila de interrupções a serem executadas, 7) interface de I/O, 8) botão de geração do código VHDL de todos os componentes.

Também pode ser visto a interface de edição na Figura 5, onde temos os seguintes componentes na interface: 1) menu de execução e controle da simulação/edição, 2) editor de código com palavras reservadas destacadas, 3) box de mensagens de erro. Já a Figura 6 ilustra o fluxo do código nos blocos do processador, com o mesmo programa usado na imagem anterior. Nessa mesma aba é possível acessar o esqueleto do código VHDL do componente clicando nele.

4.2 Simulador para o processador ARM Cortex M0®

Com o mesmo objetivo do μPD, porém trazendo uma abordagem de ensino voltada para um processador de uso comercial, foi escolhido o processador Cortex M0®, notoriamente um dos mais utilizados em projetos de sistemas embarcados. O conjunto de instruções do Cortex-M0® e a sua arquitetura foram reduzidos para simplificar o desenvolvimento, mantendo apenas o necessário para ser utilizado de forma similar ao μPD. A ferramenta, disponibilizada como uma *web app* em forma de uma SPA - *Single Page Application* -, possui dois ambientes, um destinado à simulação do μPD e outro destinado a simulação do Cortex M0®.

Sobre aspectos de arquitetura de software em que a ferramenta foi produzida, pensou-se em um desenvolvimento desacoplado, com cada serviço funcionando separadamente. Dessa forma, tem-se *endpoints* para o compilador, permitindo que o mesmo seja utilizado em outros trabalhos futuramente, para o *backend* responsável pela simulaçãod

o programa e para o *frontend*. Na Figura 7 é apresentada a arquitetura da aplicação desenvolvida e um *overview* da comunicação dos serviços.

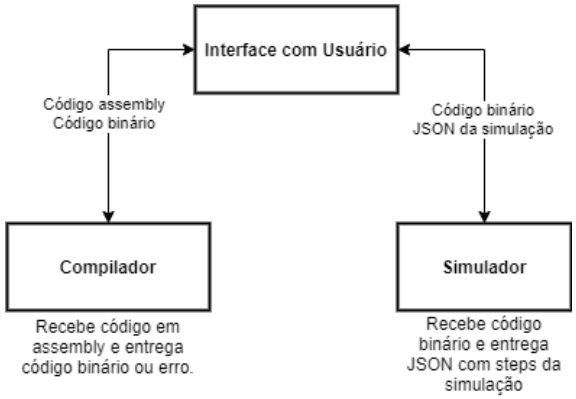


Figura 7 – Arquitetura dos serviços.

4.2.1 Aspectos da arquitetura ARM presentes na ferramenta

Os elementos do processador Cortex M0®presentes na ferramenta de simulação são reduzidos para facilitar o desenvolvimento nesta versão, podendo ser evoluídos para sua totalidade em futuros desenvolvimentos. Dessa forma, ele conta com 17 registradores - apresentados na Figura 8 -, uma memória de dados, uma memória de programa, uma memória de dispositivos, interrupções, barramento de 32 bits de escrita de dados, e barramento de 32 bits de leitura de dados. Os demais segmentos podem ser ignorados. Também está apresentada na Figura 8, a estrutura da memória, subdividida nas memórias citadas acima. Onde *Code* é a área de programa, *Interrupt* é a área de código para interrupções, SRAM a área de dados e *Peripheral* é a área de memória para acesso à dispositivos externos.

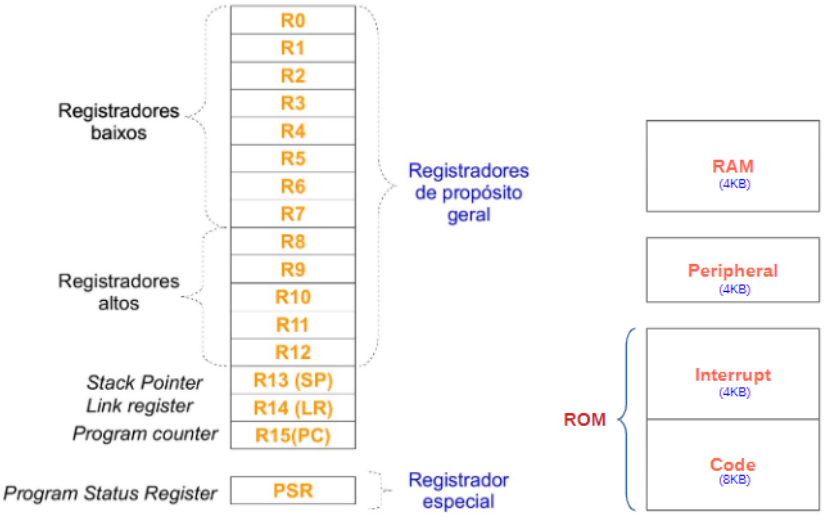


Figura 8 – Registradores e memória do Arm presentes na ferramenta.

Figura 9 – Registrador especial PSR.

O Registrador especial PSR, que está ilustrado na Figura 9, terá implementado apenas os quatro bits mais significativos do segmento ASPR (N, Z, C, V). O significado desses quatro bits (flags) são descritos a seguir:

- **N** (*Negative flag*): nível lógico 1, quando o resultado de uma operação na Unidade Lógica e a Aritmética (ULA) do processador foi um número negativo.
- **Z** (*Zero flag*): nível lógico 1, quando a operação na ULA resultou no valor zero.
- **C** (*Carry ou Borrow flag*): indica o carry, ou "vai um", no resultado de uma soma, ou borrow (empresta um) no resultado da operação de subtração.
- **V** (*Overflow flag*): nível lógico 1, quando o resultado de uma operação na ULA extrapola a quantidade de bits disponíveis no registrador do processador utilizado na operação – resultado $> [2^{31} - 1]$ ou resultado $< [-2^{31}]$.

Tabela 4 – Área de código para interrupções do Arm.

Área	Endereço inicial (hex)	Endereço final (hex)
INT(0)	2000	20FF
INT(1)	2100	21FF
INT(2)	2200	22FF
INT(3)	2300	23FF
INT(4)	2400	24FF
INT(5)	2500	25FF
INT(6)	2600	26FF
INT(7)	2700	27FF
INT(8)	2800	28FF
INT(9)	2900	29FF
INT(10)	2A00	2AFF
INT(11)	2B00	2BFF
INT(12)	2C00	2CFF
INT(13)	2D00	2DFF
INT(14)	2E00	2EFF
INT(15)	2F00	2FFF

O código relativo a cada uma das possíveis interrupções externas estarão mapeadas no segmento de memória chamado *Interrupt*, dentro da área total da ROM, sendo configuradas de acordo com a Tabela 4. Também haverá uma posição de memória para indicar em qual periférico externo o processador quer escrever, e outra posição indicando

Tabela 5 – Área de escrita/leitura em dispositivos externos do Arm.

Área	Endereço inicial (hex)	Endereço final (hex)
Periférico(0)	3000	30FF
Periférico(1)	3100	31FF
Periférico(2)	3200	32FF
Periférico(3)	3300	33FF
Periférico(4)	3400	34FF
Periférico(5)	3500	35FF
Periférico(6)	3600	36FF
Periférico(7)	3700	37FF
Periférico(8)	3800	38FF
Periférico(9)	3900	39FF
Periférico(10)	3A00	3AFF
Periférico(11)	3B00	3BFF
Periférico(12)	3C00	3CFF
Periférico(13)	3D00	3DFF
Periférico(14)	3E00	3EFF
Periférico(15)	3F00	3FFF

de qual periférico externo o processador deseja ler os dados, informação que está mapeada na tabela 5.

4.2.2 Conjunto de instruções utilizado

No ARM Cortex-M0® temos instruções e diretivas no Assembly usadas para construir programas. As diretivas são usadas para dar instruções às ferramentas de montagem, previamente à execução do código, de como tratar as diferentes partes dele. Dessa forma, nossa ferramenta utiliza as seguintes diretivas originais do processador Cortex-M0®:

- AREA: Define um bloco de código (CODE) ou dados (DATA).

  READONLY A seção é alocada em uma memória de programa (CODE).

  READWRITE A seção é alocada em uma memória de dados (DATA).

- EQU: Defini um valor numérico constante para uma palavra.

- END: Indica ofim do código fonte que contém o programa escrito em Assembly.

A palavra de dados de 32 bits é mantida, de forma que seis bits são destinados ao código da operação (OPCODE), e os demais são usados na composição da instrução. Na Figura 10 é apresentada os possíveis formatos de instrução usados na plataforma. Como foi citado anteriormente, visando a aceleração do desenvolvimento da primeira versão da ferramenta, foi adotado um conjunto de instrução reduzido, apresentado na Tabela 6.

O código binário oficial utilizado no Cortex-M0®não é aberto, portanto foi necessário criar um binário proprietário para o compilador do software de simulação. Isso não implica em problema para o aprendizado do aluno com a ferramenta, pois ainda será utilizado o Assembly oficial do processador. Esse binário fictício com o conteúdo do programa desenvolvido e compilado no editor poderá ser usado no componente da ROM do Cortex

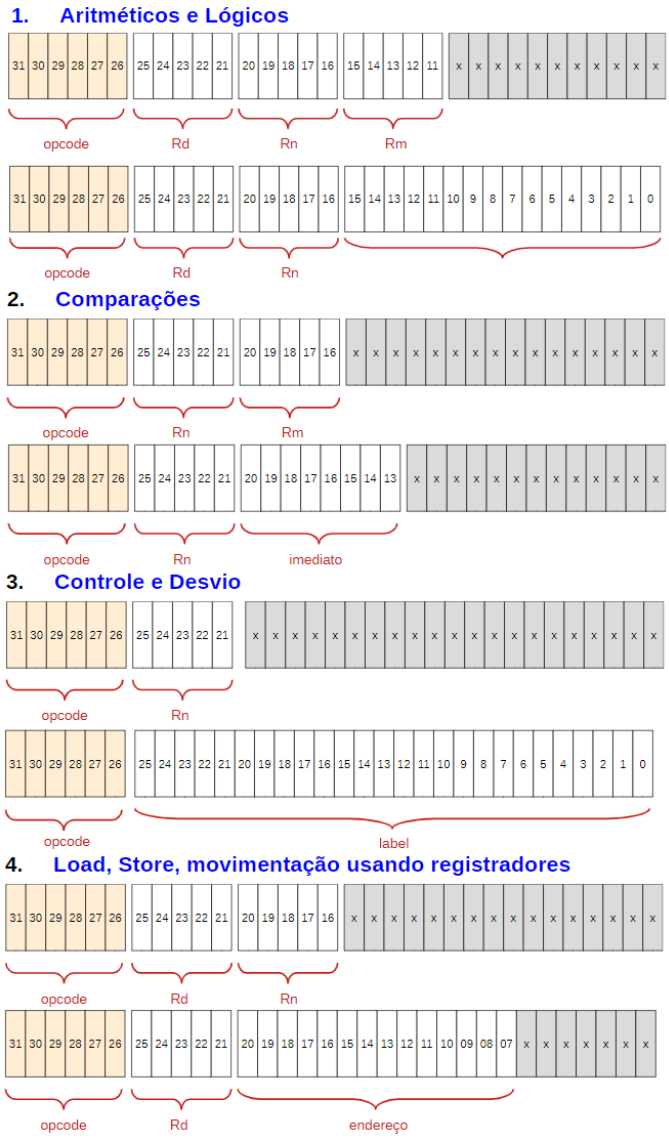


Figura 10 – Formato de instrução do Arm.

M0®em linguagem VHDL em outras ferramentas didáticas que podem ser integradas com este trabalho.

Como está apresentado na Tabela 6 e na Figura 10, a maioria das instruções possui duas formas, uma usando valores imediatos/endereçamento/label e outra usando valores providos por registradores. A operação executa a mesma função, porém a origem dos operandos são de diferentes fontes dependendo da forma utilizada, e todo o trabalho de detectar e traduzir corretamente a instrução para o binário correto fica com o compilador.

4.2.3 Interface Gráfica de Simulação

Como já mencionado anteriormente, a interface gráfica da plataforma foi desenvolvida como uma aplicação web para que seja facilmente executada em qualquer sistema operacional. Tendo duas páginas principais, uma voltada à simulação da arquitetura Arm e a outra voltada à simulação do μPD, porém com a segunda apresentando menos recursos

Tabela 6 – Instruções presentes no simulador Arm.

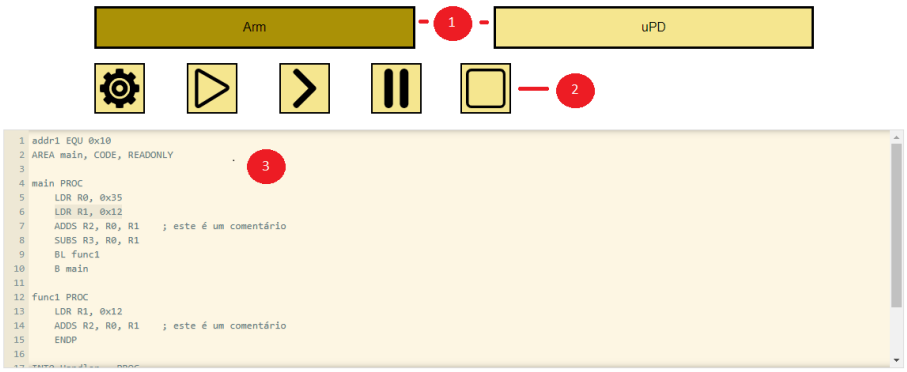
Mnemônico	OPCODE	Descrição
ADDS	000001 100001	Soma de valores e registradores.
SUBS	000010 100010	Subtração de valores e registradores.
MULS	000011 100011	Multiplicação de valores e registradores.
ANDS	000100 100100	"E"lógico de valores e registradores.
ORRS	000101 100101	"Ou"lógico de valores e registradores.
EORS	000110 100110	"Ou exclusivo"lógico de valores e registradores.
BICS	000111 100111	Zera bit.
ASRS	001000 101000	ocamento aritmético para a direita.
LSLS	001001 101001	Deslocamento lógico para a esquerda.
LSRS	001010 101010	Deslocamento lógico para a direita.
RORS	001011 101011	Rotação para a direita.
CMN	001100 101100	Compara negativo.
CMP	001101 101101	Compara.
MOVS	001110 101110	Mover.
BEQ	001111 101111	Desvio condicional se a *flag* Z é zero.
BNE	010000 110000	Desvio condicional se a *flag* Z não é zero.
BLT	010001 110001	Desvio condicional se a *flag* N é diferente da *flag* V.
BL	010010 110010	Desvio para função.
BX	010011	Retorna de uma chamada de função.
B	011011 111011	Desvio para loop.
LDR	010100 110100	Carrega registrador com valor.
STR	010101 110101	Armazena conteúdo de registrador na memória .
NOP	11111	Não faz nada.

em relação à versão inicial do μPD nesta primeira versão. Os principais componentes gráficos utilizados nas paginações de ambas arquiteturas:

- Editor de Código: utilizado para edição e criação de código de baixo nível Assembly da arquitetura escolhida. O editor inclui interface visual simplificada, a qual possui contagem de linhas escritas e suporte a algumas teclas de atalho. Na figura 11 é possível ver o editor Assembly da ferramenta com um código de exemplo.
- Menu de Execução: menu de botões usados para controlar a execução do programa, com ações como compilar, executar continuamente, executar passo a passo, pausar a execução contínua, parar a execução.
- Memória de programa: uma tabela representando a memória de programa gerada pela compilação do código, contendo o nome da área do programa, o endereço da memória, a instrução em binário e a instrução em Assembly. A instrução sendo executada no momento fica grifada em uma cor amarela.
- Memória de dados: uma tabela representando a memória de dados, com X colunas e Y linhas, que pode ser escondida com o botão ao lado esquerdo. O botão fica em uma cor amarelo mais escuro quando alguma posição da memória está sendo alterada na instrução sendo executada. Da mesma forma, a posição que está sendo alterada também fica de uma cor amarelada.
- Memória de periféricos (exclusivo Cortex M0): uma tabela representando a memória de periféricos no mesmo formato da memória de dados. O esquema de coloração segue o padrão da memória de programa.
- Menu de interrupções: um menu contendo os botões de todas as interrupções, em conjunto com uma caixa de texto que representa o endereço que está acontecendo a interrupção ao clicar.
- Banco de Registradores: uma tabela representando o banco de registradores com apenas uma coluna, que também pode ser escondida com o botão presente ao lado esquerdo. O *highlight* de cores segue o padrão de alterar quando uma instrução altera um registrador

As duas páginas possuem os mesmos recursos listados acima, porém respeitando as especifidades de cada arquitetura − número de registradores, posições de memória e etc. Na figura 11 é apresentada a interface de edição da ferramenta, onde é possível construir o código, utilizar o menu de escolha do processador e iniciar a simulação utilizando o menu de simulação. Nessa imagem é possível observar os seguintes itens: 1) menu de escolha da arquitetura simulada, 2) menu de gerenciamento de simulação, 3) editor de código, 4) memória ROM após compilar o código com sucesso.

Na figura 12 é apresentado um *snapshot* com alguns dos componentes citados acima durante a execução do programa apresentada na figura 11: 1) menu de interrupções, 2) banco de registradores, 3) memória de dados e 4) memória de dispositivos. É possível perceber a mudança de cores nos campos das memórias e do banco de registradores que estão sendo alterados pela instrução em execução, cuja também está grifada na memória de programa. Também pode-se observar que somente a interrupção 0 está disponível, pois foi a única codificada no programa compilado.



ROM

NAME	ADDRESS	BINARY INSTRUCTION	ASSEMBLY INSTRUCTION
Área de Prorgama	0x1000	11010000000000000001101010000000	LDR R0, 0x35
Área de Prorgama	0x1001	11010000001000000000100100000000	LDR R1, 0x12
Área de Prorgama	0x1002	00000100010000000000100000000000	ADDS R2, R0, R1
Área de Prorgama	0x1003	00001000011000000000100000000000	SUBS R3, R0, R1
Área de Prorgama	0x1004	11010100000000000000000000000110	BL func1
Área de Prorgama	0x1005	01101100000000000000000000000000	B main
func1	0x1006	11010000001000000000100100000000	LDR R1, 0x12
func1	0x1007	00000100010000000000100000000000	ADDS R2, R0, R1

4

Figura 11 – Tela da interface de edição da ferramenta.

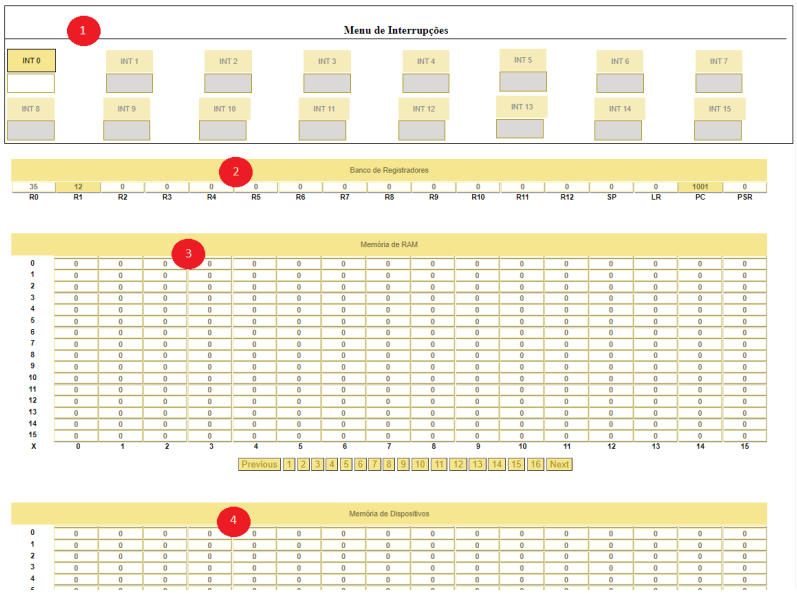


Figura 12 – Tela da interface de simulação da ferramenta.

4.3 Aspectos do μPD não presentes na Aplicação WEB

Visando um desenvolvimento de uma primeira verão dessa migração da ferramenta antiga do μPD, de aplicação *desktop* para aplicação web, alguns aspectos foram descartados neste primeiro momento. Na tabela 7 é possível ter claramente o que está presente e o que ficou para uma onda dois deste projeto.

Tabela 7 – Presença de *features* do μPD na aplicação web.

Feature	Aplicação *desktop*	Aplicação web
Editor de Código	✓	✓
Execução por texto	✓	✓
Execução por blocos	✓	-
Interrupções	✓	✓
Decimal para hexadecimal	✓	-
Geração de código VHDL	✓	-
Exemplos de código	✓	-
Instruções de Input/Output	✓	-

5 Resultados Obtidos

Considerando que este é um trabalho ainda em andamento e que não houve testes da nova ferramenta em sala de aula ainda, estão sendo considerados os resultados preliminares coletados por aplicação do μPD em sala de aula.

Existem diferentes maneiras de medir o impacto de um recurso pedagógico sobre os alunos, pode ser seguida uma abordagem de avaliação qualitativa ou quantitativa. A abordagem quantitativa requer mais dados numéricos e avalia de forma rigorosa a ferramenta, o que não faria muito sentido em uma ferramenta que ainda está em construção e precisa de *feedbacks* para evoluir. De outra forma, a avaliação qualitativa deve ser utilizada para conferir valor e fornecer feedback e não somente para atribuir nota ou conceito (CASSETTARI et al., 2001), geralmente é com base no sentimento do aluno sobre a eficácia da metodologia adotada. Como já citado em (SARTOR; SOARES; DANIEL, 2020), seguiu-se por uma abordagem qualitativa, aplicando cinco perguntas com objetivo de verificar se os alunos entenderam um conceito em comum das três disciplinas (Organização de Computadores e Arquitetura, integração em grande escala - VLSI - Projeto de Circuito e Projeto de Sistemas Embarcados): "como os dados fluem"dentro de um processador, FPGA, ou sistema embarcado usando *soft-core*.

Foram feitas até então avaliação da ferramenta com alunos das três disciplinas (Organização e Arquitetura de Computadores, Projeto de Circuito VLSI e Projeto de Sistemas Embarcados) da Universidade Federal de Santa Catarina, em dois períodos letivos. Na última aula foi aplicado um formulário de consulta com algumas perguntas aos alunos, cujas estão listadas abaixo.

1. Você entendeu o fluxo de dados dentro do processador μPD?
2. Você entendeu porque o processador μPD tem uma Memória LIFO?
3. Você entendeu a diferença entre memória de programa e memória de dados e como o processador lidar com elas?

4. Construir um bloco VHDL ajuda na compreensão de como funciona o processador?
5. Foi fácil construir um hardware embarcado usando o μPD como processador?

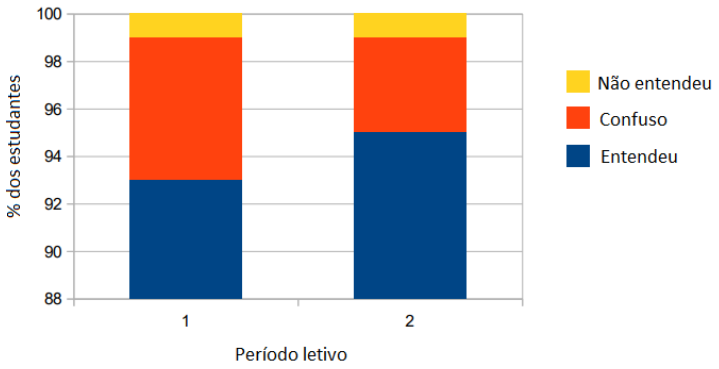


Figura 13 – Compreensão dos alunos sobre o processador μPD.

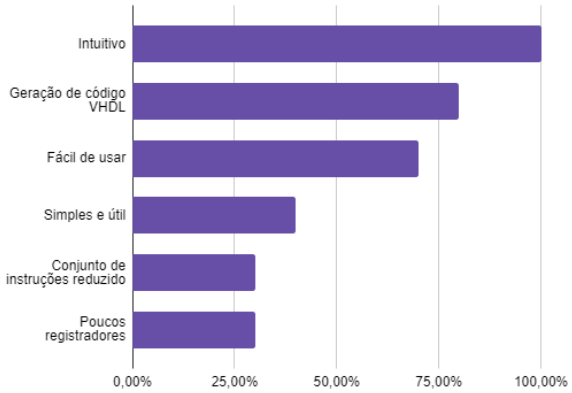


Figura 14 – Gráfico dos pontos positivos apontados pelos alunos.

O questionário foi aplicado em três turmas das disciplinas citadas acima, cada uma delas contendo vinte alunos, totalizando cento e vinte alunos ao final dos dois períodos. Os alunos de OAC responderam somente às perguntas um, dois e três; enquanto os demais alunos responderam todas elas. Na figura 13 são apresentados os resultados da pesquisa feita com os alunos, onde está sendo considerado que o aluno "entendeu"aquele que responder às cinco questões positivamente, aluno "confuso"como aquele que responde somente uma negativamente e aluno que "não entendeu"quando este responde duas ou mais questões negativamente.

Como forma de adquirir um *feedback* sobre as características da ferramenta que os alunos consideram mais importantes em seus processos de aprendizado, foi pedido para que eles apontassem uma ou mais *features* que sejam positivas. Na figura 14 são apresentados os recursos citados: "Intuitivo", "Geração de código VHDL", "Fácil de usar", "Simples e útil", "Conjunto de instruções reduzido"e "Poucos registradores". É importante destacar que a maioria dos alunos considerou a ferramenta como "Intuitivo", sendo a segunda característica mais positiva foi "Gerar código VHDL".

6 Conclusão e Trabalhos Futuros

O objetivo deste trabalho consistiu em desenvolver um simulador com características importantes para o estudo de sistemas embarcados e de tempo real, e voltado ao ensino de arquiteturas contemporâneas. Assim como foi comentado na seção 3, o projeto foi dividido em duas etapas, tendo sido desenvolvido 100% da etapa voltada à arquitetura didática onde obteve-se os resultados mencionados na seção 5.

A segunda etapa do projeto está em andamento com a evolução do software de simulação para uma arquitetura mais acessível, por ser web, e integração à arquitetura comercial. Nessa etapa o desenvolvimento se encontra em 50%, com a migração das funcionalidades da ferramenta do μPD ainda em andamento e a integração do ARM com VHDL pendente.

Os próximos passos serão focados na adição das funcionalidades faltantes do μPD, a adição da visão do caminho de dados em blocos representativos dos componentes internos da arquitetura ARM e do μPD, a integração do ARM à um kit de aprendizado em plataforma FPGA possibilitando a integração com código VHDL, melhorias no editor de código e melhorias de usabilidade no geral. Além disso, é necessário evoluir a ferramenta para que esta consiga resultados, relacionados à figura 13, de 99% de compreensão dos estudantes. Para isso é necessário continuar a introduzir a ferramenta em sala de aula e continuar coletando *feedbacks* dos alunos que fazem uso dela.

Assim como foi apresentado um comparativo das ferramentas de simulação pesquisadas na seção 2, aqui é mostrada a mesma tabela acrescida da comparação com a ferramenta proposta neste artigo (tabela 8), sua primeira versão que já foi colocada em teste em sala de aula e a versão multi arquitetura no status que ela se encontra até o momento.

A tabela ilustra que a ferramenta apresentada neste trabalho possui todos os elementos que a grande maioria dos simuladores desenvolvidos e utilizados em sala de aula, acrescida das funcionalidades cruciais para o estudo de sistemas embarcados e de tempo real. Além disso, sua evolução voltada para a web também está no mesmo caminho, trazendo uma arquitetura comercial e tornando a ferramenta de simulação multi arquitetura.

Tabela 8 – Comparativo entre as ferramentas de simulação publicadas e a primeira versão da ferramenta multi arquitetura.

	μPD	Arm+μPD	Mars	CompSim	Bipide	FRISCjs	SimuS	CESAR	RAMSES	COCONUT	ARMSim#	EduMIPS64	MarieSim
Execução passo-a-passo	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
Simulação em blocos	✓	-	-	-	✓	-	-	-	-	-	✓	-	-
Editor de código	✓	✓	✓	✓	✓	✓	✓	-	-	✓	✓	-	✓
Integração com VHDL	✓	-	-	-	-	-	-	-	-	✓	-	-	-
Interface I/O	✓	-	-	-	-	✓	✓	✓	-	-	-	-	-
Interface de Interrupção	✓	✓	-	-	-	✓	-	✓	-	-	-	-	-
Compilador/Tradutor/Montador	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
Visão do caminho de dados	✓	✓	-	-	✓	✓	-	-	-	-	-	-	✓
Baseado em acumulador	-	-	-	✓	✓	-	✓	-	-	-	-	-	✓
Baseado em registrador	✓	✓	✓	-	-	✓	-	✓	✓	✓	✓	✓	-
Multi Arquitetura	-	✓	-	-	-	-	-	-	-	-	-	-	-

Referências

BASCH DANKO ; KOVAč, M. *Osnove Processora FRISC*. [S.l.]: Antonić, 2004. Citado na página 5.

BORGES; SILVA, G. P. **SimuS-Um Simulador Para o Ensino de Arquitetura de Computadores**. *International Journal of Computer Architecture Education (IJCAE) v*, v. 5, n. 1, p. 7–12, 2016. Citado na página 5.

BORGES, J. A. S.; SILVA, G. P. Neanderwin-um simulador didático para uma arquitetura do tipo acumulador. In: *workshop sobre educação em arquitetura de computadores*. [S.l.: s.n.], 2006. Citado 2 vezes nas páginas 5 e 6.

CASSETTARI, I. S. et al. Modelo de análise qualitativa aplicado à avaliação de programas de ensino via internet. Florianópolis, SC, 2001. Citado na página 23.

CHéOUR, R. et al. Microcontrollers for iot: Optimizations, computing paradigms, and future directions. In: *2020 IEEE 6th World Forum on Internet of Things (WF-IoT)*. [S.l.: s.n.], 2020. p. 1–7. Citado na página 2.

ESMERALDO, G.; LISBOA, E. B. Compsim: Um ambiente para o ensino integrado de arquitetura e organização de computadores. In: *II Congresso sobre Tecnologias na Educação (Ctrl+ E 2017)*. [S.l.: s.n.], 2017. Citado na página 4.

HENNESSY, J. L.; PATTERSON, D. A. *Organização e Projeto de Computadores: a interface hardware/software*. [S.l.]: Elsevier Brasil, 2014. v. 4. Citado na página 4.

HORSPOOL, R. N.; LYONS, D.; SERRA, M. Armsim#-a customizable simulator for exploring the arm architecture. In: *FECS*. [S.l.: s.n.], 2009. p. 223–228. Citado na página 6.

IMPAGLIAZZO R. SLOAN, A. M. J.; SRIMANI, P. Computer engineering computing curricula. In: *Proceedings of the 34th SIGCSE Technical Symposium on Computer Science Education, Reno, Nevada, USA, February 19-23*. [S.l.: s.n.], 2003. Citado na página 9.

NULL, L.; LOBUR, J. Mariesim: The marie computer simulator. *Journal on Educational Resources in Computing (JERIC)*, ACM New York, NY, USA, v. 3, n. 2, p. 1–es, 2003. Citado na página 7.

PATTI, D. et al. Supporting undergraduate computer architecture students using a visual mips64 cpu simulator. *IEEE Transactions on Education*, IEEE, v. 55, n. 3, p. 406–411, 2012. Citado na página 7.

PEREIRA, M. C. μbip: Microcontrolador básico para o ensino de sistemas embarcados. *Trabalho de Conclusão de Curso, Ciência da Computação, Universidade do Vale do Itajaí*, 2008. Citado na página 5.

PERES, B.; ZEFERINO, C.; VIEIRA, P. Simulador web para a família de processadores bip. In: *Brazilian Symposium on Computers in Education (Simpósio Brasileiro de Informática na Educação-SBIE)*. [S.l.: s.n.], 2017. v. 28, n. 1, p. 827. Citado na página 5.

PRASAD P. W. C., A. A. A. B.; CHAN, A. Using simulators for teaching computer organization and architecture. In: *Journal of Computer Applications in Engineering Education*. [S.l.: s.n.], 2016. v. 24, n. 2. Citado na página 9.

RADIVOJEVIC, Z.; STANISAVLJEVIC, Z.; PUNT, M. Configurable simulator for computer architecture and organization. *Computer Applications in Engineering Education*, Wiley Online Library, v. 26, n. 5, p. 1711–1724, 2018. Citado na página 6.

SANTOS, R. P. dos; COSTA, H. A. X. Análise de metodologias e ambientes de ensino para algoritmos, estruturas de dados e programação aos iniciantes em computação e informática. *INFOCOMP Journal of Computer Science*, v. 5, n. 1, p. 41–50, 2006. Citado na página 3.

SARTOR, M.; SOARES, T. T. M. S.; DANIEL, M. Building a microprocessor architecture at computer engineering undergraduate courses. *International Journal of Advanced Engineering Research and Science*, v. 7, n. 7, p. 36–48, 2020. Citado na página 23.

VOLLMAR, K.; SANDERSON, P. Mars: an education-oriented mips assembly language simulator. In: *Proceedings of the 37th SIGCSE technical symposium on Computer science education*. [S.l.: s.n.], 2006. p. 239–243. Citado na página 4.

WEBER, R. F. *Fundamentos de arquitetura de computadores*. [S.l.]: Sagra Luzzatto, 2001. Citado na página 6.

ZUZAK, I. *FRISCjs: FRISC processor simulator in JavaScript*. 2016. Disponível em: <http://ivanzuzak.info/FRISCjs/webapp/>. Citado na página 5.