FACULDADE DE TECNOLOGIA SENAI “ANCHIETA”
CURSOS DE PÓS-GRADUAÇÃO EM SISTEMAS EMBARCADOS

EDUARDO ALVIM GUEDES ALCOFORADO

UTILIZAÇÃO DE FPGA NO DESENVOLVIMENTO DE UM MICROCONTROLADOR DE 32 BITS COM ARQUITETURA RISC-V E CONJUNTO DE INSTRUÇÕES RV32IM

SÃO PAULO
2022

EDUARDO ALVIM GUEDES ALCOFORADO

UTILIZAÇÃO DE FPGA NO DESENVOLVIMENTO DE UM MICROCONTROLADOR DE 32 BITS COM ARQUITETURA RISC-V E CONJUNTO DE INSTRUÇÕES RV32IM

Trabalho de Conclusão de Curso apresentado a Faculdade de Tecnologia SENAI “Anchieta” como requisito parcial para obtenção do título de Especialista em Sistemas Embarcados.

Orientador: Professor Dr. Leandro Poloni Dantas

SÃO PAULO

2022

ESTA PAGINA DEVERÁ CONTER A FICHA CATALOGRÁFICA ELABORADA PELA(S) BIBLIOTECÁRIA(S) DA INSTITUIÇÃO.

ESTE ITEM DEVERÁ SER IMPRESSO NO VERSO DA PÁGINA ANTERIOR (FOLHA DE ROSTO)

Na hora que essa ficha existir eu coloca ela no lugar disso

Obs. Essa ficha catalográfica deverá ser impressa na parte de trás da página anterior.

Eduardo Alvim Guedes Alcoforado

UTILIZAÇÃO DE FPGA NO DESENVOLVIMENTO DE UM MICROCONTROLADOR DE 32 BITS COM ARQUITETURA RISC-V E CONJUNTO DE INSTRUÇÕES RV32IM

Trabalho de Conclusão de Curso apresentado a Banca Examinadora como requisito parcial para obtenção do título de Especialista em Sistemas Embarcado.

_____

Eduardo Alvim Guedes Alcoforado

Orientador

_____

Prof. Leandro Poloni Dantas

Banca examinadora

_____

Prof. Me/Dr. XXXXXXXX

_____

Prof. Me/Dr. YYYYYYYY

São Paulo, _____ de _____2022.

AGRADECIMENTOS

Agradeço muito ao meu orientador por ter me apresentado esse tema extremamente instigante. Agradeço aos colegas e professores da época das aulas presenciais pela paciência e pelos ensinamentos preciosos que recebi nessa época (afinal, só tendo uns parafusos a menos para fazer uma pós em Sistemas Embarcados sem nunca ter estudado eletrônica antes).

Também agradeço a todos os professores que apenas conheci nas aulas on-line do curso, principalmente nas horas em que eu estava achando que não sabia nada. Realmente sou muito grato pelas conversas extra com aquelas explicações que valeram ouro.

Ao colega da 7ª Turma Diego Salviano Nagai, cujas ajudas, dicas e sugestões foram inestimáveis para esse trabalho.

E claro, também agradeço também toda equipe da Faculdade de Tecnologia SENAI “Anchieta” por todo o suporte que me foi dado quando precisei!

“Vida é o que acontece enquanto você está ocupado fazendo outros planos”.

(Autor desconhecido)

UTILIZAÇÃO DE FPGA NO DESENVOLVIMENTO DE UM MICROCONTROLADOR DE 32 BITS COM ARQUITETURA RISC-V E CONJUNTO DE INSTRUÇÕES RV32IM

Eduardo Alvim Guedes Alcoforado[1]

Orientador: Leandro Poloni Dantas[2]

RESUMO

Este trabalho tem como objetivo apresentar o desenvolvimento de um microprocessador de 32 bits, baseado na arquitetura RISC-V e que implementa o conjunto de instruções RV32IM da ISA (*Instruction Set Architecture*) RISC-V. O conjunto de instruções RV32IM contém as instruções do conjunto básico da ISA, mais as instruções referentes às operações de multiplicação e divisão de números inteiros. No tocante a microarquitetura, esta será do tipo "*single-cycle processor*", na qual a CPU (*Central Processing Unit*) irá executar uma instrução por vez e todas as instruções terão duração de um ciclo de *clock*. O desenvolvimento desse núcleo será realizado por meio da utilização de FPGAs (*Field-Programmable Gate Array*) e descrições do *harware* utilizando as HDLs (*Hardware Description Languages*) SystemVerilog e Verilog. A CPU desenvolvida aqui será testada em um kit FPGA Terasic® DE10-Lite, que possui um CI (Circuito Integrado) FPGA 10M50DAF484C7G, da família Max 10 da Altera® (atualmente Intel® *Corporation*). Em relação ao desenvolvimento em HDL da CPU será utilizada a ferramenta oficial da Altera®/Intel® para desenvolvimento em FPGA, que é o programa Quartus® Prime Lite Edition (versão 20.1.1). O trabalho conseguiu criar um núcleo de CPU RISC-V totalmente funcional, com uma memória de dados (*data memory*), uma memória de programa (*program memory*) e o hardware necessário para a implementação de uma CPU; embora a maior parte desses componentes tenha sido construída por meio de elementos lógicos, também foram utilizados outros recursos disponíveis no CI FPGA, como os blocos de memória M9K (para construir as memórias) e circuitos multiplicadores também disponíveis no CI FPGA, que tornaram a implementação da CPU mais eficiente. Por fim, para verificar o funcionamento da CPU criada aqui, foram enviados à CPU programas escritos em Assembly RISC-V e, posteriormente, verificados os resultados das operações salvos na memória de dados.

PALAVRAS-CHAVE: RISC-V. Arquitetura de Computador. Conjuntos de instruções. FPGA. Verilog/SystemVerilog.

[1]Pós-graduando em Sistemas Embarcados na Faculdade de Tecnologia SENAI Anchieta; email: dualcoforado@uol.com.br

[2]Professor doutor na Faculdade de Tecnologia SENAI Anchieta; email: leandro.poloni@senaisp.edu.br

USING FPGA IN THE DEVELOPMENT OF A 32-BIT MICROCONTROLLER WITH RISC-V ARCHITECTURE AND RV32IM INSTRUCTION SET

Eduardo Alvim Guedes Alcoforado

Leandro Poloni Dantas

ABSTRACT

The objective of this work is to present the development of a 32-bit microprocessor, based on the RISC-V architecture and which implements the ISA (Instruction Set Architecture) RISC-V instruction set RV32IM. The RV32IM instruction set contains the instructions from the ISA base set, plus instructions for integer multiplication and division operations. Regarding the microarchitecture, this will be of the “single-cycle processor” type, in which the CPU (Central Processing Unit) will execute one instruction at a time and all instructions will last one clock cycle. The development of this core will be carried out through the use of FPGAs (Field-Programmable Gate Array) and hardware descriptions using the HDLs (Hardware Description Languages) SystemVerilog and Verilog. The CPU developed here will be tested on a Terasic® DE10-Lite FPGA kit, which has an IC (Integrated Circuit) FPGA 10M50DAF484C7G, from the Max 10 family from Altera® (currently Intel® Corporation). Regarding the HDL development of the CPU, the official Altera®/Intel® tool for FPGA development will be used, which is the Quartus® Prime Lite Edition program (version 20.1.1). The work managed to create a fully functional RISC-V CPU core, with a data memory, a program memory and the hardware necessary for the implementation of a CPU; although most of these components were built using logic elements, other resources available in the FPGA IC were also used, such as the M9K memory blocks (to build the memories) and multiplier circuits also available in the FPGA IC, which made the implementation of the most efficient CPU. Finally, to verify the functioning of the CPU created here, programs written in RISC-V Assembly were sent to the CPU and, later, the results of the operations saved in the data memory were verified.

KEYWORDS: RISC-V. Instruction Set Architecture. Instruction Sets. FPGA. Verilog/SystemVerilog.

SIGLAS UTILIZADAS AQUI

ALU: *Arithmetic Logic Unit*

ARM: *Advanced RISC Machines*

ASIC: *Application-Specific Integrated Circuit*

BSD: *Berkeley Software Distribution***.**

CI: Circuito Integrado

CPU: *Central Processing Unit*

CPLD: *Complex Programmable Logic Devices*

FPGA: *Field-Programmable Grid Array*

HDL: *Hardware Description Language*

IC: *Integrated Circuit* (*Circuito Integrado*)

IP: *Intellectual Property*

ISA: *Instruction Set Architecture*

MCU: *"MicroController Unit"* ou "Microcontrolador".

RISC: *Reduced Instruction Set Computer*

RTL: *Register-Transfer-Level*

SoC*: System-on-Chip*

SoM*: System-on-Module*

SOPC: *System-On-a-Programmable Chip*

TL: *Transaction Level*

TTL: *Transistor-to-Transistor Logic*

VHDL: *VHSIC Hardware Description Language*

VHSYC: *Very High-Speed Integrated Circuits*

VLSI: *Very Large Scale Integration*

1. INTRODUÇÃO

1.1. Contexto

A arquitetura (ISA - *Instruction Set Architecture*) ou conjunto de instruções (*instruction set*) é um componente vital em qualquer processador e, sem ela, este seria incapaz de executar qualquer programa. Uma CPU (*Central Processing Unit*) somente é capaz de executar instruções de máquina, codificadas em bits e de acordo com as diretrizes da ISA dessa CPU (HOOVER, 2021). Um conjunto de instruções pode ser compreendido como o conjunto de mnemônicos associados às instruções da linguagem Assembly da ISA e que, posteriormente, são traduzidas para as instruções binárias executadas pela CPU (HOOVER, 2021; NISSAM e SCHOCKEN, 2021).

De modo geral, o conjunto de instruções implementado em uma CPU pode ser desenvolvido pelo próprio fabricante do processador ou o fabricante poderá comprar de uma empresa a IP (*Intellectual Property*) da ISA de alguma outra empresa (BAINES, 2022). Exemplos de ISAs que seguem esses moldes, merecem destaque a ISA x86 (Intel®) e ARM® (*Advanced RISC Machines*). Mas essas não são as únicas formas.

Seguindo um modelo diferente dos citados no último parágrafo, a ISA RISC-V ("*RISC Five*") vem aumentando exponencialmente a sua participação no mercado desde 2015, ano de sua fundação e vem sendo vista como uma grande inovação (URQUHART, 2021; BAILEY, 2022). O que a diferencia o RISC-V dos seus principais concorrentes é o fato de ser uma ISA de código aberto (*open-source ISA*) e isso proporciona dois diferenciais importantíssimos: 1) não há custo algum para utilizar a ISA e implementá-la em produtos novos; 2) a ISA pode ser livremente modificada e qualquer um poderá implementar modificações de acordo com suas necessidades (URQUHART, 2021; BAINES, 2022). Essa possibilidade de livre modificação da ISA é um dos principais alavancadores do RISC-V, principalmente no tocante à inovação e desenvolvimento de CPUs feitas sob medida para suas aplicações (BAILEY, 2022).

1.2. Objetivo do trabalho

O objetivo do trabalho é criar um núcleo de processador RISC-V de 32 bits que implementa um conjunto de instruções básico da ISA (RV32IM) e implementa uma microarquitetura de ciclo único (*single-cycle processor*), na qual o processador executa

uma instrução a cada ciclo de *clock* (HARRIS e HARRIS, 2022). Esse processador desenvolvido aqui se baseia no modelo proposto em Harris e Harris (2022) e, por ser algo simplificado, será referido aqui como sendo um microcontrolador ou MCU (*Microcontroller Unit*).

O MCU desenvolvido aqui implementa o conjunto de instruções básico da ISA (RV32I), que é o conjunto de instruções mínimo da ISA, mais o subconjunto de instruções RV32M, que adiciona as instruções de multiplicação, divisão e resto da divisão (WATERMAN, 2016). Embora o conjunto de instruções implementado aqui contenha apenas os recursos mínimos da ISA RISC-V, este é relativamente completo e robusto o suficiente para a implementação de aplicações básicas na área de Sistemas Embarcados (LEDIN, 2020) e bastante adequado para ser incorporado em MCUs.

O núcleo desenvolvido aqui será implementado e testado em um kit FPGA (*Field-Programmable Gate Array*) Terasic® DE10-Lite, que contém um CI (Circuito Integrado) FPGA da família MAX 10 (modelo 10M50DAF484C7G) da Altera® (atualmente Intel® Corporation). A ferramenta utilizada no desenvolvimento dessa CPU foi o software *Intel® Quartus® Prime 20.1.1 Lite Edition*. Por fim, o MCU desenvolvido aqui será escrito usando as HDLs (*Hardware Description Language*) Verilog e SystemVerilog.

1.3. RISC-V: O que é, como surgiu e missão

O conjunto de instruções é um componente intangível e não se refere a qualquer implementação física, mas sim a uma interface abstrata entre o *hardware* e o *software* de nível mais baixo (*lowest-level software*) e é formada por um conjunto de parâmetros fundamentais como conjunto de instruções, registradores, acessos de memória etc. (ASANOVIć e PATTERSON, 2014), sem os quais a CPU não é capaz de executar os programas em linguagem de máquina.

Atualmente, a maioria das ISAs disponíveis são dos tipos "**proprietária**" (*proprietary*), mas também existem algumas ISAs que adotam o formato "**licenciável**" (*licensable*) (CORDING, 2021), que é o caso das empresas ARM® e MIPS®. Quando uma ISA é do tipo proprietária, somente a empresa que desenvolveu a ISA poderá utilizá-la em seus produtos (CORDING, 2021). Já as ISAs licenciáveis são aquelas vendidas na forma de IPs (*Intellectual Properties*), onde a empresa adquirente para *royalties* para utilizar a IP em seus produtos, mas, não possui autorização para realizar modificações

nas IPs adquiridas (CORDING, 2021). Diferente desses dois casos, a ISA RISC-V é gratuita e de código completamente aberto e, portanto, qualquer pessoa pode adquirir a ISA e, também, poderá modificá-la de acordo com suas necessidades (LINUX FOUNDATION, 2021).

A ISA RISC-V se refere à quinta geração da família de arquiteturas RISC (*Reduced Instruction Set Computer*) (LINUX FOUNDATION, 2021). A arquitetura RISC nasceu em 1980, na *University of California, Berkeley* (UCB), e preconiza que um conjunto de instruções deve ser formado por apenas instruções simples e capazes de serem executadas rapidamente (PATTERSON e DITZEL, 1980). Isto é, o termo *Reduced* na sigla RISC deve ser entendido como redução na complexidade das instruções executadas pela CPU, mais simples de serem implementadas em hardware e que utilizam menos recursos da CPU (ENGHEIM, 2020). As instruções construídas sob essa filosofia RISC são otimizadas para serem utilizadas pelos compiladores e não por humanos (ENGHEIM, 2020).

O desenvolvimento do RISC-V começou em 2010, no *Parallel Computing Lab* da UCB, com um projeto interno da *UC Berkeley*, que desejava criar uma ISA para utilizar nos cursos e pesquisas ministrados pela universidade, e que sua utilização fosse livre de quaisquer restrições (ASANOVIć e PATTERSON, 2014). Desde então a ISA passou a ser utilizada não apenas dentro da UCB, mas também por pessoas de fora da universidade, o que evidenciou a importância de uma arquitetura de computador de código aberto (PATTERSON e WATERMAN, 2017), uma vez que o RISC-V, desde a sua concepção, sempre foi uma arquitetura de código aberto e distribuído sob a licença "*Creative Commons Licence*", que dá ao usuário liberdade total para usar o código, bem como modifica-lo (LINUX FOUNDATION, 2021).

O passo seguinte ocorreu em 2014 com a publicação do *white paper* de Asanović e Patterson (2014), discorrendo sobre a importância de existir uma ISA aberta e, no ano seguinte, foi criada a *RISC-V Foundation*, uma fundação sem fins lucrativos cujo objetivo é "...manter a estabilidade do RISC-V, evoluí-lo lenta e cuidadosamente, apenas por razões técnicas, e tentar torná-lo tão popular para o hardware quanto o Linux é para sistemas operacionais" (PATTERSON e WATERMAN, 2017). Com o intuito de manter a neutralidade com toda a comunidade no cenário geopolítico, em março de 2020, o nome da fundação mudou para *RISC-V International* e sua sede foi transferida para território Suíço (LINUX FOUNDATION, 2021).

Essa contextualização histórica de sucesso discorrida aqui somente se tornou realidade devido às vantagens da arquitetura RISC e do excelente projeto da arquitetura RISC-V em diversos aspectos, apresentados na próxima subseção.

1.4. A arquitetura e o conjunto de instruções RISC-V

Outra característica que torna a ISA RISC-V diferente da maioria das ISAs disponíveis é o fato desta ser uma **ISA modular** (PATTERSON e WATERMAN, 2017). Nas arquiteturas tradicionais, denominadas **ISAs Incrementais** (exemplo: x86 da Intel), sempre que a ISA é atualizada, ela é obrigada a ser compatível com as instruções antigas e, com isso, o número de instruções vai sempre aumentando a cada atualização, mesmo que instruções antigas já estejam em desuso (WATERMAN, 2016). Por causa dessa característica, o conjunto de instruções x86 possui mais de 1.500 instruções e a ISA ARM possui mais de 1.000 instruções (ENGHEIM, 2020), sendo que a mairia dessas instruções são pouquíssimo utilizadas e/ou estão em desuso (PATTERSON e WATERMAN, 2017; ENGHEIM, 2020).

Na arquitetura RISC-V, as instruções estão agrupadas em módulos (**Erro! Fonte de referência não encontrada.**). Existe um conjunto de instruções básicos, o RV32I, que jamais será alterado (PATTERSON e WATERMAN, 2017) e, portanto, qualquer programa que utilize as instruções desse módulo base será compatível tanto com versões mais antigas, quanto com versões mais novas da mesma (MARENA, 2018).

Tabela 1 – Conjunto de instruções base do RISC-V e os módulos opcionais (32 bits)

Extensão	Descrição	Num. Instruções
RV32I	Conjunto de instruções básico da ISA	47
RV32M	Multiplicação/divisão de números inteiros	8
RV32A	Operações atômicas de memória	11
RV32F	*Single-precision floating point*	26
RV32D	*Double-precision floating point*	26
RV32C	Instruções compactadas (16 bits)	36
Especificar o conjunto de instruções utilizado:		
RV32I: Para usar apenas o conjunto de instruções básico (obrigatório). **RV32IC**: RV32I + Instruções compactas de 16 bits **RV32IM**: RV32I + multiplicação/divisão de inteiros. **RV32IMF**: RV32IM + *Single-Precision* **RV32IMD**: RV32IM + *Double-Precision* **RV32IMFD**: RV32IM + *Single-Precision* + *Double-Precision*		

Fonte: Kanter (2016) e Waterman (2016)

O conjunto RV32I é constituído por apenas 47 instruções (ENGHEIM, 2020), e se referem apenas de operações básicas de processamento e de operações aritméticas de

adição e subtração de números inteiros (WATERMAN, 2016). Caso seja identificada a necessidade de utilização de operações de multiplicação e/ou divisão, usando números inteiros, basta realizar a operação de carregamento de instruções RV32IM, que carrega as instruções do conjunto base (RV32I) e as instruções de multiplicação e divisão (RV32M) (PATTERSON e HENESSY, 2021). Da mesma forma, havendo uma necessidade de utilização de números decimais basta carregar o conjunto de instruções usando RV32IMF (para precisão simples) ou RV32IMD (para precisão dupla).

O RISC-V também possui uma extensão com instruções compactadas (RV32C), que contém instruções de 16 bits e, com isso, é possível gerar linhas de instruções com 32 bits, mas que executam duas instruções compactadas (WATERMAN, 2016). Pra utilizar essas instruções, carregar o conjunto de instrução RV32IC.

Além das extensões listadas no **Erro! Fonte de referência não encontrada.**, também existe o conjunto de instruções RV32E, que contém as instruções do conjunto RV32I, mas utiliza apenas 16 registradores (WATERMAN, 2016). Segundo Wateman (2016) o componente estrutural mais caro de um núcleo RISC-V é essa estrutura com 31 registradores. Sendo assim, em aplicações embarcadas em que não seja necessário a utilização de todos esses registradores, o conjunto básico RV32E oferece uma versão menos custosa (WATERMAN, 2016).

Essa estrutura modular do RISC-V é um dos principais diferenciais e motivos de sucesso da ISA, uma vez que possibilita a criação de processadores especializados para as aplicações que se destinam (URQUHART, 2021; BAYLEY, 2022; SPERLING, 2022). Uma evidência dessa característica como um fator crítico de sucesso da ISA se refere à grande adoção desta em aplicações de computação heterogênea, que se baseia na utilização de hardware especializado para a realização de diferentes atividades (ENGHEIM, 2022), como o desenvolvimento de CPUs dedicadas a operações de *Machine Learn* (aprendizado de máquina) e de microcontroladores (MCUs) (BAYLEY, 2022; MOORE, 2022). Engheim (2022) cita como possível exemplo disso a criação de SoCs compostos por vários núcleos de CPU RISC-V, cada um especializado em um tipo de processamento.

Essa possibilidade de criação de núcleos especializados reduz os custos de fabricação dos chips, uma vez que este é proporcional à área (*die area*) do chip (ENGHEIM, 2022). Isto é, ao produzir um chip formado apenas pelo hardware necessário para sua a aplicação, a área desse chip será menor e, com isso, será possível obter mais

chips a partir de um único *wafer* (PATTERSON e WATERMAN, 2017) e, consequentemente, reduzir o custo unitário de cada chip produzido.

A próxima subseção apresenta mais sobre a adoção do RISC-V em pesquisas acadêmicas, bem como na criação de chips comerciais.

1.5. Posicionamento do RISC-V no mercado

O RISC-V é amplamente estudado no ambiente acadêmico, principalmente em instituições que recebem financiamento público e que não desejam desperdiçar recursos com projetos que lhes prendam a um produto específico, principalmente em casos em que esses produtos são fornecidos por empresas do tipo *for-profit*, que fornecem produtos com algum nível de restrição (TURLEY, 2020). Entretanto, a arquitetura também é bastante adotada em CPUs comerciais, produzidas por empresas como Espressif, SiFive e Sipeed.

A arquitetura RISC-V foi muito bem recebida desde o princípio. No mesmo ano que a *RISC-V Foundation* foi criada, a fundação se associou a três grandes empresas americanas de tecnologia: Google, HP e Oracle (MERRIT, 2015). Outra grande empresa que também se associou nesses primeiros anos foi a Western Digital, que em 2019 lançou o núcleo *SweRV* como sendo completamente de código aberto (HRUSKA, 2019; WESTERN DIGITAL, 2019). Essa chegada ao mercado com uma proposta inovadora também fez do RISC-V alvo de diversos ataques por parte da ARM, inclusive de um site criado por esta para difamar as arquiteturas de código aberto (HRUSKA, 2018; ENGHEIM, 2020).

Nesses primeiros anos de existência, houve pouco interesse de adoção do RISC-V pelas empresas chinesas, uma vez a arquitetura ARM tinha um domínio muito bem estabelecido entre os fabricantes desse país (EE TIMES, 2018). Entretanto, esse cenário vem mudando desde meados de 2019 quando os fabricantes chineses passaram a sofrer sanções dos principais fornecedores de tecnologia de processadores como a Intel e ARM (HRUSKA, 2019). Desde então diversos fabricantes chineses, como a *AllWinner*, passaram a desenvolver seus próprios núcleos de CPU usando a arquitetura RISC-V (EE TIMES, 2021).

Também visando diminuir a dependência de empresas como Intel e ARM, o governo russo investiu em 2021 30 bilhões de Rublos em uma parceria entre três empresas russas, com o objetivo de produzir novas CPUs RISC-V para serem usadas

em laptops e servidores do próprio governo e, com isso, reduzir a dependência de tecnologias estrangeiras nessa área (HRUSKA, 2021) .

Por fim, o RISC-V tem ganhado ainda mais visibilidade nos anos de 2021 e 2022 por causa da Intel. Em 2021 a Intel estabeleceu uma parceria com a SiFive, uma das principais fabricantes de CPUs baseadas em RISC-V, com a criação da IFS (*Intel Foundry Services*), oferecendo à SiFive a produção de CPUs RISC-V utilizando o processo de 7nm (HRUSKA, 2021; HRUSKA, 2021). Já em 2022, a SiFive conseguiu levantar US$175 milhões em fundos para investir no desenvolvimento de novos chips RISC-V com desempenho superior aos chips da ARM e, também, planeja futuramente realizar um IPO (*Initial Public Offering*) para abrir o capital da empresa no mercado financeiro (DAHAD, 2022). Já a Intel, anunciou as suas novidades em fevereiro de 2022, que foi o anúncio de um fundo de desenvolvimento bilionário, voltado ao desenvolvimento de novas CPUs RISC-V, para serem fabricadas nas plantas da IFS (HRUSKA , 2022). O segundo anúncio da Intel nesse mesmo mês foi de que a empresa planeja licenciar CPUs híbridas que combinam os recursos das arquiteturas ARM, x86 e RISC-V (HRUSKA , 2022).

1.6. Sobre as próximas seções do trabalho

Dando prosseguimento ao trabalho, a próxima seção discorre sobre alguns aspectos importantes para o desenvolvimento do trabalho, que são as instruções da ISA RISC-V, estrutura do processador implementado aqui e um pouco sobre FPGAs e linguagens de descrição de hardware (HDLs – *Hardware Description Languages*). A terceira seção apresenta a implementação do processador no kit FPGA e a quarta seção encerra o trabalho.

2. DESENVOLVIMENTO

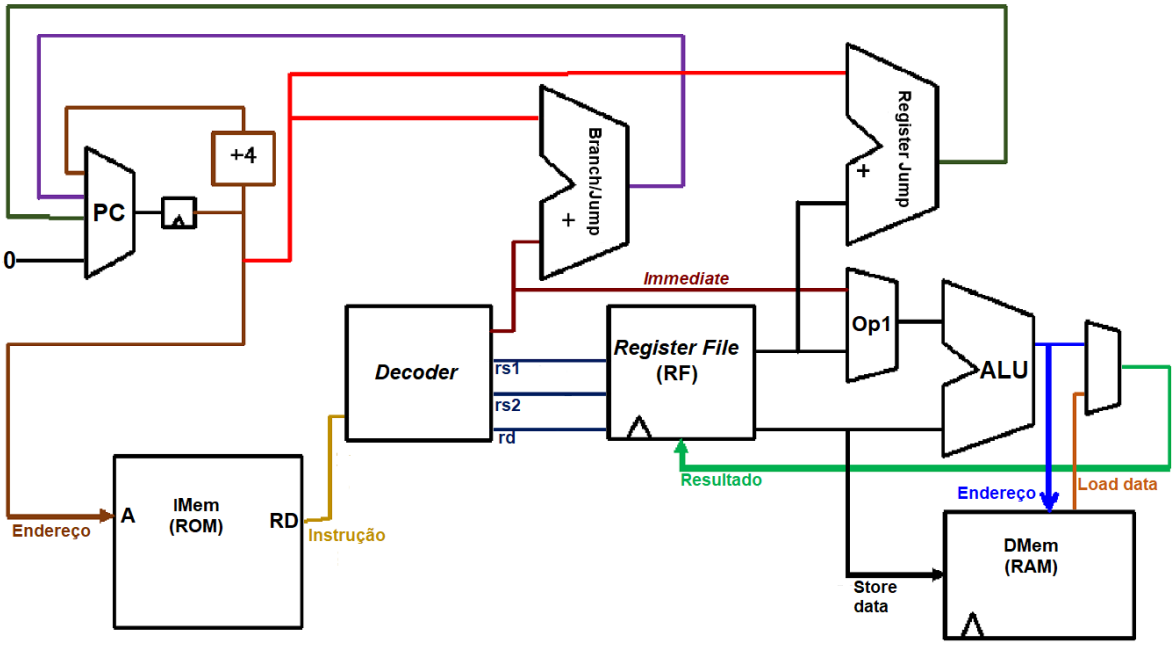
Essa seção apresenta o desenvolvimento do microcontrolador RV32IM com microarquitetura *single-cycle*, baseado no conteúdo apresentado em Harris e Harris (2022). A primeira subseção descreve os componentes fundamentais de um processador e explica como são codificadas as instruções do Assembly RISC-V; a segunda apresenta o hardware e software utilizado nos testes e a terceira explica as HDLs utilizadas nos projetos, Verilog e SystemVerilog. Por fim, a quarta subseção apresenta as instruções

implementadas no MCU analisado e o repositório com os projetos implementados nos testes realizados aqui.

2.1. Componentes Fundamentais de uma CPU e Formato das Instruções do Conjunto RV32IM

A Figura 1 apresenta o diagrama esquemático básico do MCU RISC-V baseado na ISA RV32IM e desenvolvido aqui. O MCU possui 33 registradores, um registrador cujo valor é sempre 0 (`x0`), 31 registradores de uso geral (registradores `x1-x31`) e um registrador para armazenar o endereço da próxima instrução a ser executada (registrador `pc`) (WATERMAN, 2016). Esses registradores são utilizados para armazenar os dados que serão utilizados nas operações das instruções e, também, valores utilizados com muita frequência (PATTERSON e WATERMAN, 2017). Os registradores do *Register File* (RF) são consideradas as memórias mais rápidas de qualquer CPU (NISSAM e SCHOCKEN, 2021; HARRIS e HARRIS, 2022). No caso do MCU desse trabalho, os 33 registradores do RF possuem 32 bits de comprimento.

Figura 1 – Diagrama de blocos do processador RV32I desenvolvido aqui



Fonte: Material do curso on-line “Building a RISC-V CPU Core” (Hoover, 2021) – Modificado

Como mostra a Figura 1, o RF não é a única memória disponível no MCU. Este possui outras duas memórias, com maior capacidade de armazenamento, mas com acessos mais lentos (HARRIS e HARRIS, 2022); uma memória do tipo ROM (*Read-Only Memory*) – Bloco “IMem” na Figura 1 – para armazenar o programa em execução

(*program memory*), e uma memória do tipo RAM (*Random-Access Memory*) – Bloco “DMem” na Figura 1 – para o armazenamento temporário de dados usados ao longo do programa executado (HARRIS e HARRIS, 2022).

Outro componente importante é o decodificador de instruções – bloco “*Decoder*” na Figura 1 – que é responsável por identificar o formato da instrução, os operandos e outros campos desta (HOOVER, 2021; HARRIS e HARRIS, 2022). Como mostra a Tabela 2, o conjunto de instruções RV32I contém 47 instruções, subdivididas nos 6 tipos indicados na Tabela 2 (WATERMAN, 2016). Já o subconjunto RV32M contém mais 8 instruções e todas elas seguem o formato *R-Type* apresentado na primeira linha da Tabela 2 (PATTERSON e WATERMAN, 2017).

Tabela 2 – Formato das instruções do subconjunto de instruções RV32IM

Tipo	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R-Type	funct7							rs2					rs1					funct3			rd					OpCode						
I-Type	imm[11:0]												rs1					funct3			rd					OpCode						
S-Type	imm[11:5]							rs2					rs1					funct3			imm[4:0]					OpCode						
B-Type	{ imm[12], imm[10:5] }							rs2					rs1					funct3			{ imm[4:1], imm[11] }					OpCode						
U-Type	imm[31:12]																				rd					OpCode						
J-Type	{ imm[20], imm[10:1], imm[11], imm[19:12] }																				rd					OpCode						

Observações:

rd: Registrador de destino; é o registrador que recebe o resultado da instrução executada.

rs1: Registrador com o operando 1; é o registrador que armazena o valor utilizado no primeiro operando da instrução executada.

rs2: Registrador com o operando 2; é o registrador que armazena o valor utilizado no segundo operando da instrução executada.

imm: Valor da constante passada para a instrução.

OpCode: Os bits 0 e 1 (opcode[1:0]) são iguais a 1, para todas as instruções do subconjunto RV32I.

funct7: Esse parâmetro somente é utilizado nas instruções do subconjunto *R-Rtype*.

Fonte: RISC-V INTERNATIONAL (2022)

Quanto às instruções decodificadas em “*decoder*”, é importante ressaltar aqui que TODAS as instruções armazenadas em “IMem” possuem, invariavelmente, 32 bits de comprimento (RISC-V INTERNATIONAL, 2022).

Outro bloco vital do MCU é o bloco “ALU” na Figura 1. O bloco “ALU” possui duas entradas que recebem os valores dos operandos da operação que será realizada pela instrução (HOOVER, 2021). Como mostra a Tabela 2, os valores das entradas de “ALU” podem ser dois valores armazenados em registradores (instruções no formato *R-Type*) (LEDIN, 2020; HOOVER, 2021) ou um valor armazenado em registrador e outro um valor imediato, ou “*immediate*” (formato *I-Type*) que é o caso dos formatos (LEDIN, 2020; HOOVER, 2021). Quanto as instruções que utilizam os outros formatos apresentados na Tabela 2, a maioria não utiliza a ALU.

No tocante ao valor imediato utilizado por alguns formatos de instruções da Tabela 2, esse valor é codificado de maneira diferente para cada tipo. A Tabela 3 mostra como é gerado o valor imediato de 32 bits para cada um dos tipos de instruções vistos na Tabela 2.

Tabela 3 – Formação do campo “*immediate*” das instruções

	imm[31:0]																													
Tipo	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
I-Type	i[31]	i[31]	i[31]	i[31]	i[31]	i[31]	i[31]	i[31]	i[31]	i[31]	i[31]	i[31]	i[31]	i[31]	i[31]	i[31]	i[31]	i[31]	i[31]	i[31]	i[31]	i[30]	i[29]	i[28]	i[27]	i[26]	i[25]	i[24]	i[23]	i[22]	i[21]	i[20]
S-Type	i[31]	i[31]	i[31]	i[31]	i[31]	i[31]	i[31]	i[31]	i[31]	i[31]	i[31]	i[31]	i[31]	i[31]	i[31]	i[31]	i[31]	i[31]	i[31]	i[31]	i[31]	i[30]	i[29]	i[28]	i[27]	i[26]	i[25]	i[11]	i[10]	i[9]	i[8]	i[7]
B-Type	i[31]	i[31]	i[31]	i[31]	i[31]	i[31]	i[31]	i[31]	i[31]	i[31]	i[31]	i[31]	i[31]	i[31]	i[31]	i[31]	i[31]	i[31]	i[31]	i[31]	i[7]	i[30]	i[29]	i[28]	i[27]	i[26]	i[25]	i[11]	i[10]	i[9]	i[8]	0
U-Type	i[31]	i[30]	i[29]	i[28]	i[27]	i[26]	i[25]	i[24]	i[23]	i[22]	i[21]	i[20]	i[19]	i[18]	i[17]	i[16]	i[15]	i[14]	i[13]	i[12]	0	0	0	0	0	0	0	0	0	0	0	0
J-Type	i[31]	i[31]	i[31]	i[31]	i[31]	i[31]	i[31]	i[31]	i[31]	i[31]	i[31]	i[31]	i[19]	i[18]	i[17]	i[16]	i[15]	i[14]	i[13]	i[12]	i[20]	i[30]	i[29]	i[28]	i[27]	i[26]	i[25]	i[24]	i[23]	i[22]	i[21]	0
R-Type	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Fonte: RISC-V INTERNATIONAL (2022)

Por fim, no tocante ao registrador `pc`, o valor em sua saída pode ser um de 4 valores possíveis: 1) valor 0, caso a entrada `reset` seja ativada; 2) valor atual mais 4, quando instruído para executar a próxima instrução de “*IMem*”; 3) valor atual, mais *Branch*, caso seja recebida uma instrução de *branching* e a condição seja verdadeira; e 4) valor atual, mais deslocamento, caso tenha recebido uma instrução de salto incondicional (`jal` ou `jalr`) (HOOVER, 2021; HARRIS e HARRIS, 2022).

A próxima subseção apresenta o hardware e o software utilizados nos projetos.

2.2. Hardware e software utilizados

Os projetos que implementam a CPU RV32IM foram desenvolvidos para serem executados no kit FPGA Terasic DE10-Lite. Esse kit possui um CI FPGA Intel®/Altera® MAX 10 10M50DAF484C7G, o qual inclui alguns recursos muito importantes para o projeto desenvolvido nesse trabalho.

Dentre os recursos de maior interesse do CI FPGA 10M50DAF484C7G, este possui 49.760 elementos lógicos (LEs – *Logic Elements*), 288 blocos multiplicadores de 9 bits (*embedded multiplier 9-bit elements*), 1.677.312 bits distribuídos em 182 blocos de memória M9K (9.216 bits por bloco M9K) e suporte para até 360 pinos GPIO (TERASIC, 2020).

Nos projetos desenvolvidos aqui, os blocos M9K serão utilizados para criar tanto a memória de programa (ROM), quanto a memória de dados (RAM). A utilização fornece ao projeto uma memória rápida e que não utiliza os elementos lógicos do CI FPGA (CHU, 2012). Já os multiplicadores de 9 bits, estes são hardware específicos para operações de multiplicação, cuja utilização também reduz o uso de elementos lógicos (CHU, 2012).

Os projetos analisados mais adiante foram todos criados usando a ferramenta de desenvolvimento oficial da Altera®/Intel®, o software *Quartus® Prime Lite Edition*, versão 20.1.1. Quanto aos módulos do projeto, estes foram escritos em Verilog (memórias RAM e ROM e o Conjunto de Registradores) e SystemVerilog (todos os outros módulos).

O *Quartus® Prime Lite Edition* foi usado também para a realização dos testes dos projetos que executam scripts em Assembly RISC-V. No caso desses testes, é

necessário primeiro enviar o projeto compilado para o CI FPGA e, posteriormente, verificar os resultados na ferramenta "*In-System Memory Content Editor*" do Quartus.

A subseção a seguir apresenta uma breve história das linguagens Verilog e SystemVerilog.

2.3. Sobre as linguagens Verilog e SystemVerilog

A linguagem Verilog foi criada pela Gateway Design Automation em 1984, e era uma linguagem proprietária desta empresa (HARRIS e HARRIS, 2022). Em 1989 a Gateway foi comprada pela Cadence e, em 1990, a linguagem se tornou aberta (HARRIS e HARRIS, 2022). Por fim, desde 1995 a linguagem Verilog passou a ser normatizada pela IEEE (*Institute of Electrical and Electronics Engineers*), com o pronunciamento 1364-1995 e, posteriormente, atualizada em 2001 e em 2005 (IEEE Std. 1364-2001[1]).

Já a linguagem SystemVerilog é uma linguagem derivada do Verilog, com a adição de novas funcionalidades para esta. Ela também é normatizada pelo IEEE desde 2005 (IEEE Std. 1800-2005) e a versão atual do pronunciamento que a normatiza é o IEEE Std. 1800-2009[2]. Conforme já mencionado, o software *Quartus® Prime* possui suporte a ambas as linguagens.

A próxima subseção apresenta as instruções do conjunto RV32IM implementadas no projeto e sobre o repositório onde os projetos estão disponíveis.

2.4. Instruções Implementadas e Repositório com os projetos

A Tabela 4 apresenta as instruções do conjunto RV32IM que foram implementadas no núcleo de processamento desenvolvido aqui.

[1] Disponível em: https://ieeexplore.ieee.org/document/1620780

[2] Disponível em: https://ieeexplore.ieee.org/document/5354441

Tabela 4 – Instruções de Assembly RISC-V do conjunto de instruções RV32IM

Tipo	Subgrupo	Formato	Instruções do grupo
R-Type	**Aritmética**	`<instr> rd, rs1, rs2`	`add sub mul mulh mulhsu mulhu div divu rem remu`
	Lógica	`<instr> rd, rs1, rs2`	`xor or and`
	Deslocamento	`<instr> rd, rs1, rs2`	`sll srl sra`
	Menor ou Igual	`<instr> rd, rs1, rs2`	`slt sltu`
S-Type	***Store***	`<instr> rs1, imm(rs2)`	`sw`
B-Type	***Branch***	`<instr> rd, rs1, imm`	`beq bne blt bge bltu bgeu`
U-Type	***Load Data***	`<instr> rd, rs1`	`lui auipc`
J-Type	***Jump-and-Link***	`<instr> rd, imm`	`jal`
I-Type	***Jump-and-link Reg***	`<instr> rd, imm(rs1)`	`jalr`
	Aritmética	`<instr> rd, rs1, imm`	`addi`
	Lógica	`<instr> rd, rs1, imm`	`xori ori andi`
	Deslocamento	`<instr> rd, rs1, imm`	`slli srli srai`
	Menor ou Igual	`<instr> rd, rs1, imm`	`slti sltiu`
	Load Data	`<instr> rd, rs1, imm`	`lw`

Fonte: Patterson e Henessy (2021); Harris e Harris (2022)

Foi criado um repositório no Github com todos os projetos usados nos testes desse trabalho[3]. Todos os projetos desse repositório foram testados no kit FPGA DE10-Lite e todos funcionaram corretamente. A próxima seção apresenta mais detalhes sobre os testes realizados para verificar o funcionamento do MCU desenvolvido aqui.

3. METODOLOGIA

A primeira subseção apresenta o conteúdo do repositório com o material desse trabalho (https://github.com/dualvim/Repo_TCC_SistEmb), a segunda apresenta e explica o programa em Assembly RISC-V usado no primeiro teste do projeto da pasta `Proj_RV32IM_03_De10Lite`, enquanto a terceira subseção explica o segundo script usado no teste do projeto `Proj_RV32IM_03_De10Lite`.

3.1. Repositório com os arquivos do projeto

O primeiro projeto do repositório, pasta `Proj_RV32I_01_ALU_RISC-V`, contém um projeto que realiza uma depuração manual da ALU. Os códigos das instruções são selecionados por meio dos switches do kit DE10-Lite e o resultado das operações aparece nos displays de sete segmentos. O valor do primeiro operando da ALU é 23 e o valor do segundo operando é 3. O resultado dos testes dessa pasta estão disponíveis nos vídeos dentro da pasta.

[3] Repositório: https://github.com/dualvim/Repo_TCC_SistEmb

O projeto da pasta `Proj_RV32IM_02_De10Lite` contém uma subversão do projeto final, onde o módulo principal é testado no módulo criado no arquivo `testbench.sv`. O projeto dessa pasta é testado na ferramenta ModelSim®, que acompanha o software Quartus® Prime.

Por fim, o projeto final está na pasta `Proj_RV32IM_03_De10Lite`, neste são enviados para oi CI FPGA o projeto e um arquivo compilado com o programa testado. Os programas compilados realizam diversas operações da CPU e, então, salvam os resultados na memória RAM da CPU. O conteúdo da memória RAM fica disponível através da ferramenta "*In-System Memory Content Editor*" do Quartus® Prime, após enviar o projeto compilado para o FPGA.

A próxima subseção apresenta o primeiro programa em Assembly RISC-V uado nos testes.

3.2. Programa em Assembly RISC-V com o primeiro teste

O Quadro 1 apresenta um código em Assembly para realizar uma soma de 0 a 9, armazenando o resultado dessa soma no registrador `x2`. Depois, o valor salvo em `x2` é copiado para o registrador `x6` e, em seguida, é subtraído 44 do valor salvo em `x6` e o resultado é salvo no registrador `x7`. Por fim, os valores dos registradores `x1` a `x6` são salvos na memória RAM.

Quadro 1 – Script de teste 1: Soma dos valores 1 a 9

```
# Inicio
addi x1, x0, 4
addi x2, x0, 0     # x2 = 0 + 0
addi x3, x0, 10    # x3 = 0 + 10
addi x4, x0, 1     # x13 = 0 + 1
addi x5, x0, 0
# Bloco 'loop'
loop:
add x2, x2, x4      # x14 = x13 + x14
addi x4, x4, 1      # x4 = x4 + 1
sub x5, x3, x4
beq x5, x0, label2
beq x4, x4, loop    # Se x4 < x3, voltar 2 linhas
# Linhas executadas após a última execução do bloco 'loop'
label2:
addi x6, x0, 0
add x6, x0, x2      # Copiar para x6 o valor em x2
addi x7, x0, 0
addi x7, x6, -44    # x7 = x6 - 44
# Salvar os dados na memoria
sw   x1, 0(x1)
lw  x10, 0(x1)
sw   x2, 4(x1)
lw  x10, 4(x1)
sw   x3, 8(x1)
lw  x10, 8(x1)
sw   x4, 12(x1)
lw  x10, 12(x1)
sw   x7, 20(x1)
lw  x10, 20(x1)
sw   x6, 24(x1)
```

```
lw  x10, 24(x1)
# Bloco 'end'
end: beq  x0, x0, end     # Encerra o programa
```

Fonte: Hoover (2021)

A subseção a seguir apresenta o programa em Assembly do segundo teste.

3.3. Programa em Assembly RISC-V com o segundo teste

O Quadro 2 apresenta o programa usado no segundo teste. O programa testa as instruções `addi`, `add`, `sub`, `and`, `or`, `xor`, `sll`, `srl`, `sra`, `mul`, `div` e `rem` e escreve os resultados dessas operações na memória RAM.

Quadro 2 – Script de teste 2: Instruções diversas

```
# Registradores com os valores usados nas operacoes
addi x12, x0, 23   # val 1
addi x13, x0, 3    # val 2
# Salvar os valores dos registradores
sw   x12, 4(x0)
lw   x3,  4(x0)
sw   x13, 8(x0)
lw   x3,  8(x0)
# Soma
add x2, x12, x13
sw  x2, 12(x0)
lw  x3, 12(x0)
# subtracao
sub x2, x12, x13
sw  x2, 16(x0)
lw  x3, 16(x0)
# and
and x2, x12, x13
sw  x2, 20(x0)
lw  x3, 20(x0)
# or
or x2, x12, x13
sw x2, 24(x0)
lw x3, 24(x0)
# xor
xor x2, x12, x13
sw  x2, 28(x0)
lw  x3, 28(x0)
# sll
sll x2, x12, x13
sw  x2, 32(x0)
lw  x3, 32(x0)
# srl
srl x2, x12, x13
sw x2, 36(x0)
lw x3, 36(x0)
# slt
slt x2, x12, x13
sw x2, 40(x0)
lw x3, 40(x0)
# sra
sra x2, x12, x13
sw  x2, 44(x0)
lw  x3, 44(x0)
# mul
mul x2, x12, x13
sw  x2, 48(x0)
lw  x3, 48(x0)
# mulh
mulh x2, x12, x13
sw   x2, 52(x0)
lw   x3, 52(x0)
# div
div x2, x12, x13
sw  x2, 56(x0)
```

```
lw  x3, 56(x0)
# rem
rem x2, x12, x13
sw  x2, 60(x0)
lw  x3, 60(x0)
# Bloco 'end'
end: beq x0, x0, end     # Encerra o programa
```

Fonte: Patterson e Henessy (2021) e Harris e Harris (2022)

Com essas explicações sobre os programas testados, a próxima seção apresenta os resultados dos testes apresentados nessa seção.

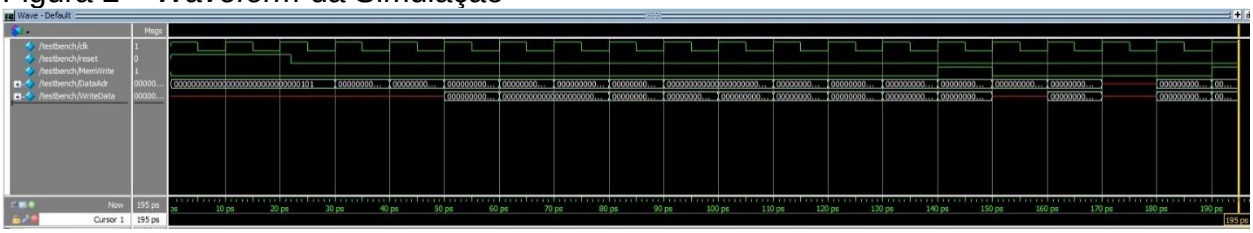
4. RESULTADOS E/OU DISCUSSÕES

A primeira subseção apresenta os testes do projeto da pasta `Proj_RV32IM_02_De10Lite`. A segunda e a terceira, apresentam os testes no projeto da pasta `Proj_RV32IM_03_De10Lite`. A última subseção apresenta os diagramas gerados no *Quartus® Prime Lite Edition* 20.1.1 com os diagramas esquemáticos do *hardware* criado aqui.

4.1. Resultado do *Waveform* da Simulação (projeto da pasta `Proj_RV32IM_02_De10Lite`)

O projeto da pasta `Proj_RV32IM_02_De10Lite` é uma versão ampliada do projeto de CPU RV32I *Single-Cycle* apresentado em Harris e Harris (2022). O resultado da simulação desse projeto ampliado está na Figura 2, indicando que o conteúdo adicionado ao exemplo inicial está correto.

Figura 2 – *Waveform* da Simulação



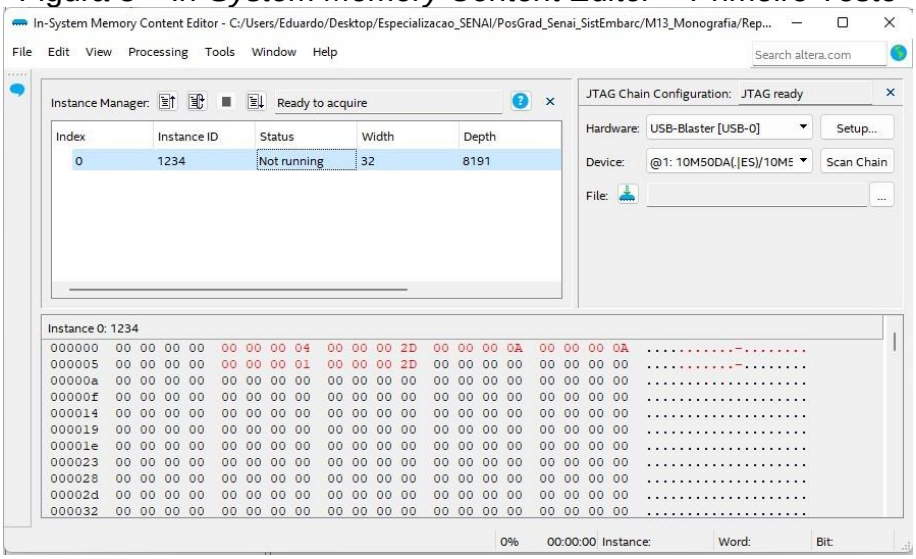
Fonte: Saída gerada pelo ModelSim®

Com os resultados da Figura 2 confirmados, a próxima etapa foi testar os programas em Assembly RISC-V criados para esse trabalho.

4.2. Projeto `Proj_RV32IM_03_De10Lite`, primeiro teste

Nesse primeiro teste, foi testado o programa que realiza a soma dos números de 0 a 9. O resultado esperado dessa operação é 45 e esse valor deverá ser salvo nos endereços de memória `0x08` e `0x28`. Os resultados apresentados na Figura 3 confirmam que o valor da soma realizada foi 45 (`0x2D`) e, também, o valor dessa soma foi salvo nos endereços de memória corretos.

Figura 3 – *In-System Memory Content Editor* – Primeiro Teste



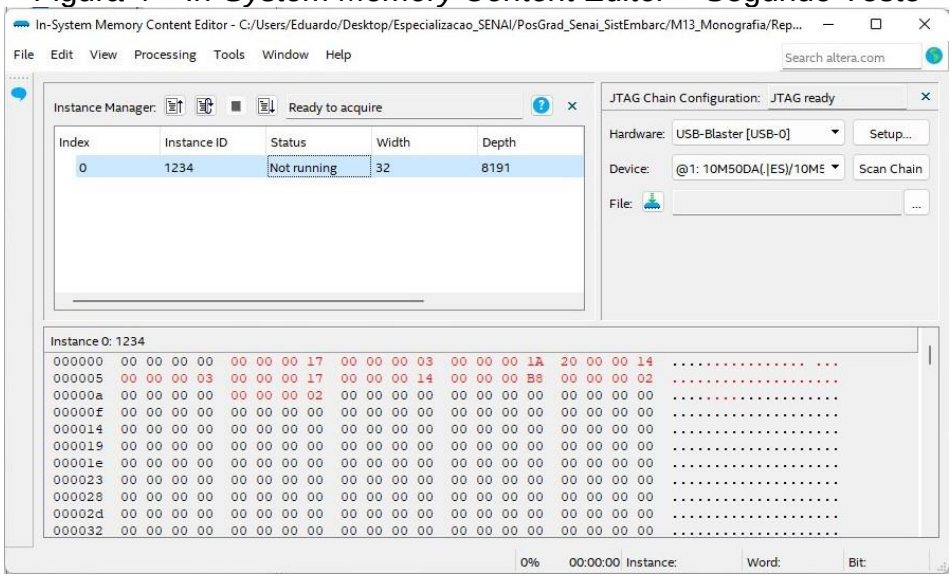
Fonte: Captura de tela do *Quartus Prime 20.1.1 Lite Edition*

Esse primeiro programa apenas testa algumas poucas instruções elementares disponíveis no conjunto RV32IM. A próxima subseção realiza um teste mais abrangente, testando um número maior de instruções diferentes.

4.3. Projeto `Proj_RV32IM_03_De10Lite`, segundo teste

Os resultados da realização dos testes do segundo programa de teste estão apresentados na Figura 4.

Figura 4 – *In-System Memory Content Editor* – Segundo Teste



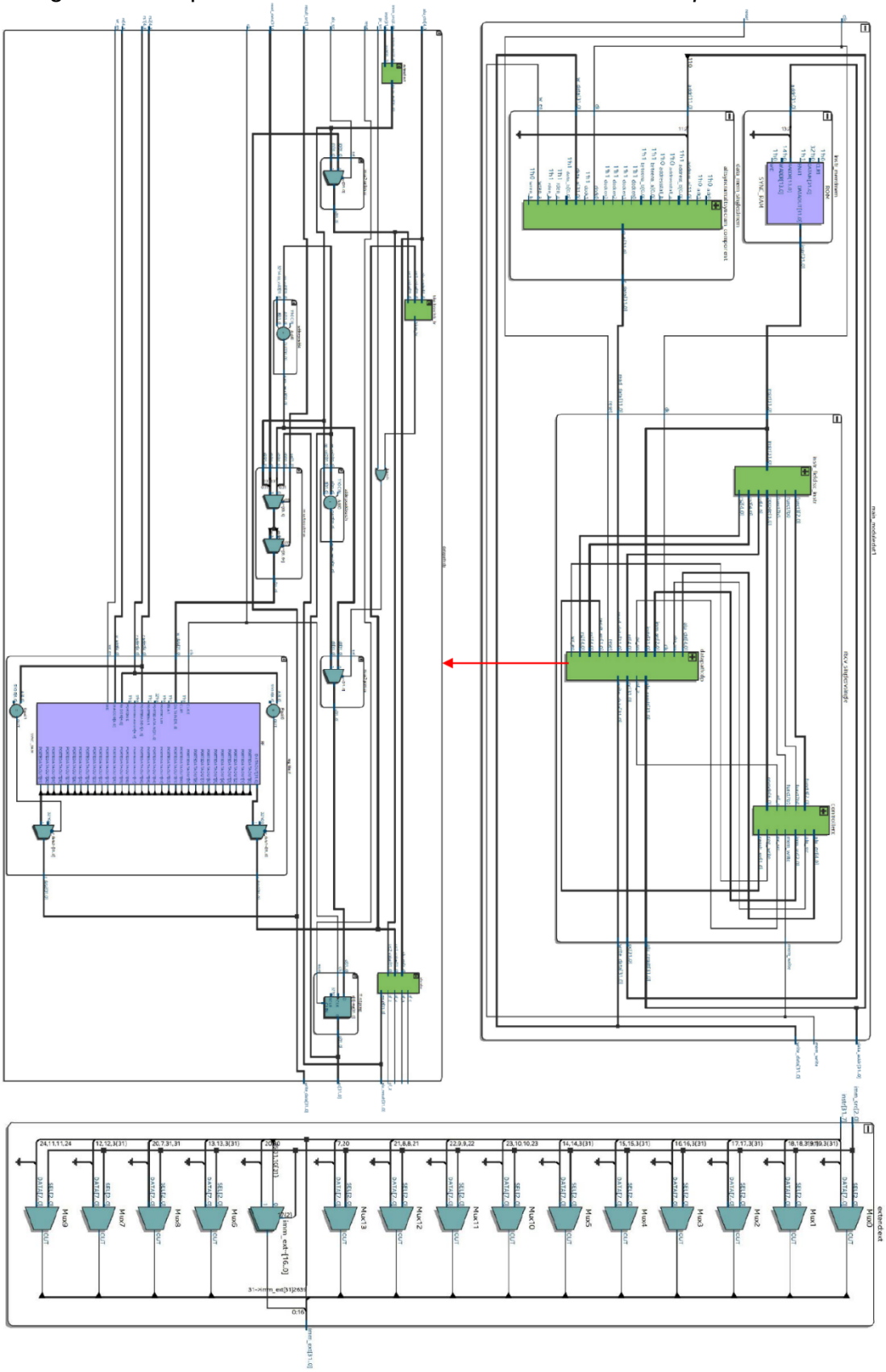
Fonte: Captura de tela do *Quartus Prime 20.1.1 Lite Edition*

Assim como ocorreu nos testes da subseção anterior, os resultados armazenados na RAM foram os mesmos que eram esperados.

Por fim, a próxima subseção apresenta os diagramas gerados no *Quartus® Prime Lite Edition*, apresentando os diagramas esquemáticos do hardware analisado aqui.

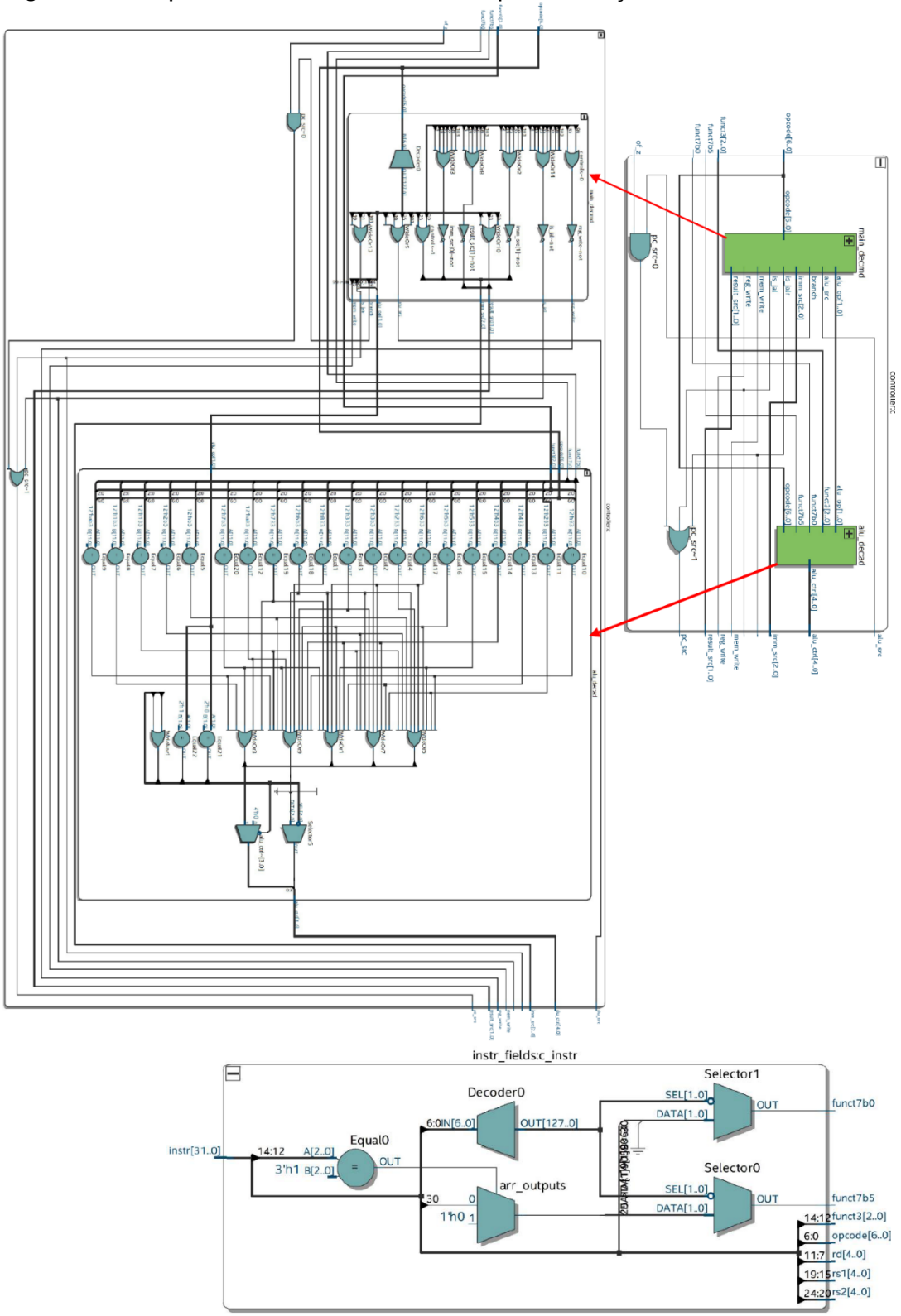
4.4. Diagramas com os esquemáticos da CPU e os principais submódulos

Figura 5 – Esquemático: Núcleo do MCU, Memórias, *Datapath* e *extend*.



Fonte: Diagrama gerado no *Quartus Prime 20.1.1 Lite Edition*

Figura 6 – Esquemático: *Controller* e Campos Instrução



Fonte: Diagrama gerado no *Quartus Prime 20.1.1 Lite Edition*

5. CONCLUSÃO

O trabalho apresentou o desenvolvimento de um MCU (microcontrolador) RISC-V de 32 bits que implementa o conjunto de instruções básico da ISA, mais as operações de divisão e multiplicação (conjunto de instruções RV32IM). O núcleo desenvolvido possui uma microarquitetura do tipo "*single-cycle*", que executa uma única instrução a cada ciclo de *clock*, algo razoavelmente adequado considerando o contexto de um microcontrolador.

O hardware utilizado para implementar o MCU foi um FPGA com CI da família Altera®/Intel® MAX 10 (no caso aqui, foi o CI FPGA 10M50DAF484C7G). Os CIs FPGA dessa família inclui alguns recursos importantes de hardware, como os blocos de memória M9K e os multiplicadores embarcados (*embedded multiplier*), cuja utilização melhora a eficiência da utilização dos elementos lógicos disponíveis no CI FPGA (CHU, 2012).

A verificação das operações da CPU consistiu em enviar scripts em Assembly RISC-V para a CPU e mandar salvar os resultados das operações na memória RAM da CPU. Os resultados das operações salvos na RAM foram verificado com a ferramenta "*In-System Memory Content Editor*" do software Quartus Prime Lite Edition. Os resultados salvos na memória RAM bateram com os resultados esperados ddas operações.

Sendo assim, o presente trabalho desenvolveu um microcontrolador baseado na ISA RISC-V completamente funcional, capaz de executar as instruções do conjunto RV32IM. Para trabalhos futuros, sugere-se o desenvolvimento de versões mais robustas da CPU, como a inclusão de pipelines, de modo a melhorar o desempenho da CPU criada aqui.

6. REFERÊNCIAS

ASANOVIć, K.; PATTERSON, D. A. **Instruction Sets Should Be Free: The Case For RISC-V**. UC Berkeley. Berkeley, p. 7. 2014. Disponivel em: <https://www2.eecs.berkeley.edu/Pubs/TechRpts/2014/EECS-2014-146.pdf>.

BAILEY, B. Why RISC-V Is Succeeding. **Semiconductor Engineering**, 24 abr. 2022. Disponivel em: <https://semiengineering.com/why-risc-v-is-succeeding/>.

BAINES, R. Differentiation And Architecture Licenses In RISC-V, 26 maio 2022. Disponivel em: <https://semiengineering.com/differentiation-and-architecture-licenses-in-risc%e2%80%91v/>.

BAYLEY, B. Why RISC-V Is Succeeding. **Semiconductor Engineering**, 24 abr. 2022. Disponivel em: <https://semiengineering.com/why-risc-v-is-succeeding/>.

CHU, P. P. **Embedded SOPC Design with NIOS II Processor and Verilog Examples**. Hoboken: John Wiley & Sons, Inc., 2012. ISBN: ISBN 978-1-118-01103-4.

CORDING, S. What Is RISC-V? An In-Depth introduction to the RISC-V Instruction Set Architecture. **Elektor Magazine**, 2021. Disponivel em: <https://www.elektormagazine.com/articles/what-is-risc-v>.

DAHAD, N. SiFive Raises $175M To Quicken ‘Arm Intercept’ Strategy. **EE Times**, 16 mar. 2022. Disponivel em: <https://www.eetimes.com/sifive-raises-175m-to-quicken-arm-intercept-strategy/>.

EE TIMES. Why RISC-V Lags in China. **EE Times**, 15 nov. 2018. Disponivel em: <https://www.eetimes.com/why-risc-v-lags-in-china/>.

EE TIMES. Allwinner launches the first RISC-V application processor. **EE Times**, 15 abr. 2021. Disponivel em: <https://www.eetimes.com/allwinner-launches-the-first-risc-v-application-processor/>.

ENGHEIM, E. What Does RISC and CISC Mean in 2020? **Medium**, 27 jul. 2020. Disponivel em: <https://medium.com/swlh/what-does-risc-and-cisc-mean-in-2020-7b4d42c9a9de>.

ENGHEIM, E. What Is Innovative About RISC-V? **Medium**, 24 dez. 2020. Disponivel em: <https://medium.com/swlh/what-is-innovative-about-risc-v-a821036a1568>.

ENGHEIM, E. RISC-V Is The King of Heterogenous Computing. **Medium**, 06 jan. 2022. Disponivel em: <https://erik-engheim.medium.com/risc-v-the-king-of-heterogenous-computing-11b47649e691>.

ENGHEIM, E. The Case for RISC-V on Desktops and Servers. **Medium**, 06 jan. 2022. Disponivel em: <https://erik-engheim.medium.com/the-case-for-risc-v-on-desktops-and-severs-60b0106c636b>.

HARRIS, S. L.; HARRIS, D. M. **Digital Design and Computer Architecture - RISC-V Edition**. Cambridge, MA: Morgan Kaufmann, 2022. ISBN: 978-0-12-820064-3.

HOOVER, S. Building a RISC-V CPU Core. **Linux Foundation/edX**, 2021. Disponivel em: <https://www.edx.org/course/building-a-risc-v-cpu-core>.

HRUSKA , J. Intel Announces Billion-Dollar Development Fund, Boosts RISC-V Processors. **ExtremeTech**, 09 fev. 2022. Disponivel em:

<https://www.extremetech.com/extreme/331461-intel-announces-billion-dollar-development-fund-boosts-risc-v-processors>.

HRUSKA , J. Intel Plans to License Hybrid Chips That Combine ARM, RISC-V, and x86. **ExtremeTech**, 2022. Disponivel em: <https://www.extremetech.com/computing/331740-intel-plans-to-license-cores-that-combine-arm-risc-v-and-x86>.

HRUSKA, J. ARM Kills Its RISC-V FUD Website After Staff Revolt. **ExtremeTech**, 12 ago. 2018. Disponivel em: <https://www.extremetech.com/computing/273236-arm-kills-its-risc-v-fud-website-after-staff-revolt>.

HRUSKA, J. Cut Off From ARM, x86, What CPU Architectures Can Huawei Use? **ExtremeTech**, 23 maio 2019. Disponivel em: <https://www.extremetech.com/computing/291875-cut-off-from-arm-x86-what-cpu-architectures-can-huawei-actually-use>.

HRUSKA, J. Western Digital's RISC-V 'Swerv' Core Now Available for Free. **ExtremeTech**, 15 fev. 2019. Disponivel em: <https://www.extremetech.com/computing/285856-western-digitals-risc-v-swerv-core-now-available-for-free>.

HRUSKA, J. Intel Will Offer SiFive RISC-V CPUs on 7nm, Plans Own Dev Platform. **ExtremeTech**, 24 jun. 2021. Disponivel em: <https://www.extremetech.com/computing/324075-intel-will-offer-sifive-risc-v-cpus-on-7nm-plans-own-dev-platform>.

HRUSKA, J. Rumor: Intel May Buy RISC-V CPU Designer SiFive to Fend off ARM. **ExtremeTech**, 11 jun. 2021. Disponivel em: <https://www.extremetech.com/computing/323647-rumor-intel-may-buy-risc-v-cpu-designer-sifive-to-fend-off-arm>.

HRUSKA, J. Russia to Build 8-Core RISC-V CPUs for Laptops, Government Systems. **ExtremeTech**, 15 jul. 2021. Disponivel em: <https://www.extremetech.com/computing/324735-russia-to-build-8-core-risc-v-cpus-for-laptops-government-systems>.

LEDIN, J. **Modern Computer Architecture and Organization:** Learn x86, ARM, and RISC-V architectures and the design of smartphones, PCs, and cloud servers. Birmingham: Packt Publishing, 2020. ISBN: 978-1-83898-439-7.

LINUX FOUNDATION. **Introduction to RISC-V**. [S.l.]: [s.n.], 2021. Disponivel em: <https://www.edx.org/course/introduction-to-risc-v>.

MARENA, T. 11 Myths About the RISC-V ISA. **Electronic Design**, 31 jan. 2018. Disponivel em: <https://www.electronicdesign.com/technologies/embedded-revolution/article/21806096/11-myths-about-the-riscv-isa>.

MERRIT, R. Google, HP, Oracle Join RISC-V. **EE Times**, 28 dez. 2015. Disponivel em: <https://www.eetimes.com/google-hp-oracle-join-risc-v/>.

MOORE, S. K. RISC-V AI Chips Will Be Everywhere. **IEEE Spectrum**, 24 fev. 2022. Disponivel em: <https://spectrum.ieee.org/risc-v-ai>.

NISSAM, N.; SCHOCKEN, S. **The Elements of Computing Systems**. 2a. ed. Cambridge: The MIT Press, 2021. ISBN: 9780262539807.

PATTERSON, D. A.; DITZEL, D. R. The Case for the Reduced Instruction Set Computer. **ACM SIGARCH Computer Architecture News**, 8, n. 6, 1980, 25–33. DOI: 10.1145/641914.641917.

PATTERSON, D. A.; HENESSY, J. L. **Computer Organization and Design - RISC-V Edition**. 2a. ed. Cambridge: Morgan Kaufmann, 2021. ISBN: 978-0-12-820331-6.

PATTERSON, D.; WATERMAN, A. S. **Guia Prático RISC-V:** Atlas de uma arquitetura aberta. 1a. ed. San Francisco: Strawberry Canyon LLC, 2017. ISBN: 978-0-9992491-1-6.

RISC-V INTERNATIONAL. **The RISC-V Instruction Set Manual - Unprivileged ISA**. Berkeley: [s.n.], 2022. Disponivel em: <https://riscv.org/technical/specifications/>.

SPERLING, E. Which Processor Is Best? **Semiconductor Engineering**, 01 mar. 2022. Disponivel em: <https://semiengineering.com/which-processor-is-best/>.

TERASIC. **DE10-Lite User Manual**. [S.l.]: [s.n.], 2020.

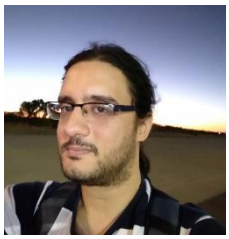
TURLEY, J. Why Universities Want RISC-V. **EE Journal**, 27 out. 2020. Disponivel em: <https://www.eejournal.com/article/why-universities-want-risc-v/>.

URQUHART. Is RISC-V The Future? **Semiconductor Engineering**, 29 jul. 2021. Disponivel em: <https://semiengineering.com/is-risc-v-the-future/>.

WATERMAN, A. S. **Design of the RISC-V Instruction Set Architecture**. University of California, Berkeley. Berkeley, p. 117. 2016. Disponivel em: <https://www2.eecs.berkeley.edu/Pubs/TechRpts/2016/EECS-2016-1.pdf>.

WESTERN DIGITAL. **RISC-V and Open Source Hardware Address New Compute Requirements**. Western Digital. San Jose, p. 8. 2019. Disponivel em: <https://documents.westerndigital.com/content/dam/doc-library/en_us/assets/public/western-digital/collateral/tech-brief/tech-brief-western-digital-risc-v.pdf>.

7. SOBRE OS AUTORES



Eduardo Alvim Guedes Alcoforado
Currículo Lattes: http://lattes.cnpq.br/0205554239317512

Leandro Poloni Dantas (Orientador)
Currículo Lattes:
https://www.escavador.com/sobre/7253202/leandro-poloni-dantas