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

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

Eduardo Alvim Guedes Alcoforado[1], [i]
Leandro Poloni Dantas[2], [ii]
Marcones Cleber Brito da Silva[3], [iii]
Luis Carlos Canno[4], [iv]
Fernando Simplicio de Sousa[5], [v]

Data de submissão: (19/05/23) Data de aprovação: (24/07/23)

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. O conjunto de instruções RV32IM contém as instruções do conjunto básico, RV32I, mais as instruções referentes às operações de multiplicação e divisão de números inteiros. A microarquitetura implementada é a "*single-cycle processor*", em que é executada uma instrução por vez, cada uma executada em um ciclo de *clock*. O núcleo foi desenvolvido por meio da utilização de FPGAs e das linguagens de descrição de hardware SystemVerilog e Verilog. A CPU desenvolvida aqui foi testada em um CI (circuito integrado) FPGA da família Max 10 da Altera® (atualmente Intel® Corporation). Em relação ao ambiente de desenvolvimento da CPU, foi utilizada a ferramenta oficial da Altera®/Intel® para desenvolvimento em FPGA, o programa *Quartus® Prime Lite Edition*. Neste trabalho, foi criado um núcleo de CPU RISC-V totalmente funcional, com uma memória de dados, uma memória de programa, além do hardware necessário para a implementação da CPU; embora a maior parte desses componentes tenha sido construída por meio de elementos lógicos, também foram utilizados outros recursos adicionais disponíveis no CI FPGA da Intel, como os blocos de memória M9K (para construir as memórias) e circuitos multiplicadores, que tornaram a implementação da CPU mais eficiente. Por fim, o MCU desenvolvido nesse trabalho oferece uma implementação completa de um núcleo de processamento que poderá ser incorporado a novos MCUs e, também, oferece contribuições práticas às pesquisas de desenvolvimento usando FPGAs.

Palavras-chave: RISC-V; arquitetura do conjunto de instruções; FPGA; Verilog; sistemas embarcados.

[1] Pós-graduado na Faculdade de Tecnologia SENAI São Paulo – Campus "Anchieta". E-mail: dualcoforado@uol.com.br
[2] Professor Dr. na Faculdade de Tecnologia SENAI São Paulo – Campus "Anchieta". E-mail: leandro.poloni@sp.senai.br
[3] Professor Me. na Faculdade de Tecnologia SENAI São Paulo – Campus "Anchieta". E-mail: marcones.silva@sp.senai.br
[4] Professor Esp. na Faculdade de Tecnologia SENAI São Paulo – Campus "Anchieta". E-mail: luis.canno@sp.senai.br
[5] Professor Me. na Faculdade de Tecnologia SENAI São Paulo – Campus "Anchieta". E-mail: fernando.simplicio@sp.senai.br

ABSTRACT

This work aims to present the development of a 32-bit microprocessor, based on the RISC-V architecture and which implements the RV32IM instruction set, which contains the instructions from the basic set, RV32I, plus instructions for multiplying and dividing integers. The microarchitecture implemented is the “single-cycle processor”, which executes one instruction at a time, and each one executed in one clock cycle. This core was developed using FPGAs and the SystemVerilog and Verilog hardware description languages. The CPU developed here was tested on an Altera® (currently Intel® Corporation) FPGA IC (integrated circuit) from the Max 10 family. Regarding the development environment, it was used the official Altera®/Intel® tool for FPGA development, the Quartus® Prime Lite Edition software. In this work, a fully functional RISC-V CPU core was created, with a data memory, a program memory, in addition to the necessary hardware for the CPU implementation; although most of these components were built using logic elements, other additional resources available in Intel's FPGA IC were also used, such as the M9K memory blocks (to build the memories) and multiplier circuits, which made the CPU implementation more efficient. Finally, the MCU developed in this work offers a complete implementation of a processing core that can be incorporated into new MCUs and also offers practical contributions to development research using FPGAs.

Keywords: RISC-V; instruction set architecture; FPGA; Verilog; embedded systems.

1 INTRODUÇÃO

A arquitetura do conjunto de instruções (*Instruction Set Architecture*, ISA) é um componente vital em qualquer processador e, sem ela, este seria incapaz de executar qualquer programa. Uma unidade central de processanto (*central processing unit*, CPU) somente é capaz de executar instruções de máquina, codificadas em bits e de acordo com os formatos estruturados das instruções executadas por esta 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 em 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 outra empresa desenvolvedora uma propriedade intelectual (*Intellectual Property*, IP) com a ISA já pronta (BAINES, 2022). Exemplos de arquiteturas que são comercializadas na forma de IPs e que merecem destaque são as ISAs x86 (Intel®) e ARM®. No entanto, essas não são as únicas formas possíveis de se implementar um conjunto de instruções.

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 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 fatores de alavancagem do RISC-V, principalmente no tocante à inovação

e desenvolvimento de CPUs feitas sob medida para aplicações específicas (BAILEY, 2022).

1.1 Objetivo do trabalho

O objetivo deste trabalho foi criar um núcleo de processador RISC-V de 32 bits que implementa o conjunto de instruções básico da ISA, chamado de RV32I e o subconjunto de instruções RV32M. Esse conjunto utiliza uma microarquitetura em que todas as instruções são do tipo ciclo único (*single-cycle processor*); isto é, todas as instruções do processador são executadas em um único ciclo de clock, e o processador executa apenas uma instrução por vez (HARRIS e HARRIS, 2022). O núcleo desenvolvido baseia-se no modelo mais simples proposto em Harris e Harris (2022). Por ser um dispositivo simplificado, este é referido aqui como sendo um microcontrolador ou MCU (*microcontroller unit*).

1.2 Justificativa

Embora a MCU implementada inclua o conjunto básico de instruções, também foi acrescido o subconjunto RV32M, que adiciona as instruções de multiplicação, divisão e resto da divisão (WATERMAN, 2016). Apesar do conjunto de instruções implementado contenha apenas os recursos mínimos da ISA RISC-V, esse conjunto básico é considerado abrangente e robusto o suficiente para suportar a implementação de aplicações fundamentais na área de Sistemas Embarcados (LEDIN, 2020), sendo altamente adequado para ser incorporado em MCUs.

O trabalho está dividido em mais quatro seções. A segunda seção apresenta a arquitetura RISC-V e a sua história, apresenta a fundação teórica de um núcleo básico de processador e as instruções do conjunto RV32IM do núcleo criado aqui. A terceira seção apresenta as linguagens Verilog e SystemVerilog, o procedimento de desenvolvimento da CPU RV32IM e os programas em *Assembly* RISC-V usados para testar a CPU criada. A quarta seção apresenta os resultados dos testes da CPU criada e a última subseção apresenta os diagramas esquemáticos do núcleo desenvolvido. Por fim, a última seção apresenta a conclusão do trabalho.

2 REVISÃO DE LITERATURA

A seguir, é fornecida uma revisão da literatura que aborda a arquitetura RISC-V. Iniciamos com uma contextualização histórica, seguida pela apresentação da arquitetura e do conjunto de instruções do RISC-V. Posteriormente, é discutida a posição da ISA no mercado e sua aceitação. Por fim, são apresentados os componentes básicos que compõem uma CPU e explicada a estrutura das instruções do RISC-V.

2.1 RISC-V: o que é, como surgiu e missão

O conjunto de instruções é um componente intangível e não se refere a uma implementação física, mas sim a uma interface abstrata entre o hardware e o software de nível mais baixo (*lowest-level software*); essa interface 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 do tipo proprietárias (*proprietary*), mas também existem algumas ISAs que adotam o formato licenciável (*licensable*) (CORDING, 2021), que é o caso das ISAs comercializadas pelas 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, onde a empresa adquirente para *royalties* para utilizar a IP em seus produtos, mas, não possui autorização para realizar quaisquer modificações nessas 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 essa ISA e, também, poderá modificá-la de acordo com suas necessidades (LINUX FOUNDATION, 2021).

O RISC-V se refere à quinta geração da família de arquiteturas de computadores com conjunto reduzido de instruções (*Reduced Instruction Set Computer*, RISC) (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, 2020a). As instruções construídas sob essa filosofia RISC são otimizadas para serem utilizadas pelos compiladores e não por humanos (ENGHEIM, 2020a).

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 modificá-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 a Suíça (LINUX FOUNDATION, 2021).

2.2 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 essas 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, 2020b), sendo que a maioria dessas instruções são pouquíssimo utilizadas e/ou estão em desuso (PATTERSON e WATERMAN, 2017; ENGHEIM, 2020b).

Na arquitetura RISC-V, as instruções estão agrupadas em módulos (Tabela 1). 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 do RISC-V (MARENA, 2018).

O conjunto RV32I é constituído por apenas 47 instruções (ENGHEIM, 2020b), 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) de 16 bits. Para utilizar essas instruções, é necessário carregar o conjunto de instrução RV32IC.

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).

Além das extensões listadas na Tabela 1, também existe o conjunto de instruções RV32E, que contém as instruções do conjunto RV32I, mas utiliza apenas 16 registradores (WATERMAN, 2016). Segundo Waterman (2016) o componente estrutural mais caro de um núcleo RISC-V é essa estrutura com 32 registradores. Sendo assim, em aplicações embarcadas em que não seja necessária 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; BAILEY, 2022; SPERLING, 2022). Uma evidência dessa característica como um fator crítico de sucesso da ISA se refere à grande adoção do RISC-V em aplicações de computação heterogênea, que se baseia na utilização de hardware especializado para a realização de diferentes atividades (ENGHEIM, 2022a), como o desenvolvimento de CPUs dedicadas à operações de aprendizado de máquina (*machine learn*) e de microcontroladores (MCUs) (BAILEY, 2022; MOORE, 2022). Engheim (2022) cita como possível exemplo disso a criação de sistema em um único chip (*System-on-a-Chip*, SoC) composto 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 custo é proporcional à área do chip (*die area*) (ENGHEIM, 2022b). 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.

2.3 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 visam lucro), 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, que é inteiramente de código aberto (HRUSKA, 2019b; WESTERN DIGITAL, 2019). Essa chegada ao mercado como sendo uma proposta inovadora também fez do RISC-V alvo de diversos ataques por parte de empresas como a ARM®, que criou um site para difamar as arquiteturas de código aberto como o RISC-V (HRUSKA, 2018; ENGHEIM, 2020b).

Nesses primeiros anos de existência, houve pouco interesse de adoção do RISC-V pelas empresas chinesas, uma vez que 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, 2019a). Desde então diversos fabricantes chineses, como a AllWinner, passaram a desenvolver seus próprios núcleos de CPU usando a arquitetura RISC-V (ALLWINNER, 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 suas próprias CPUs RISC-V para serem usadas em laptops e servidores

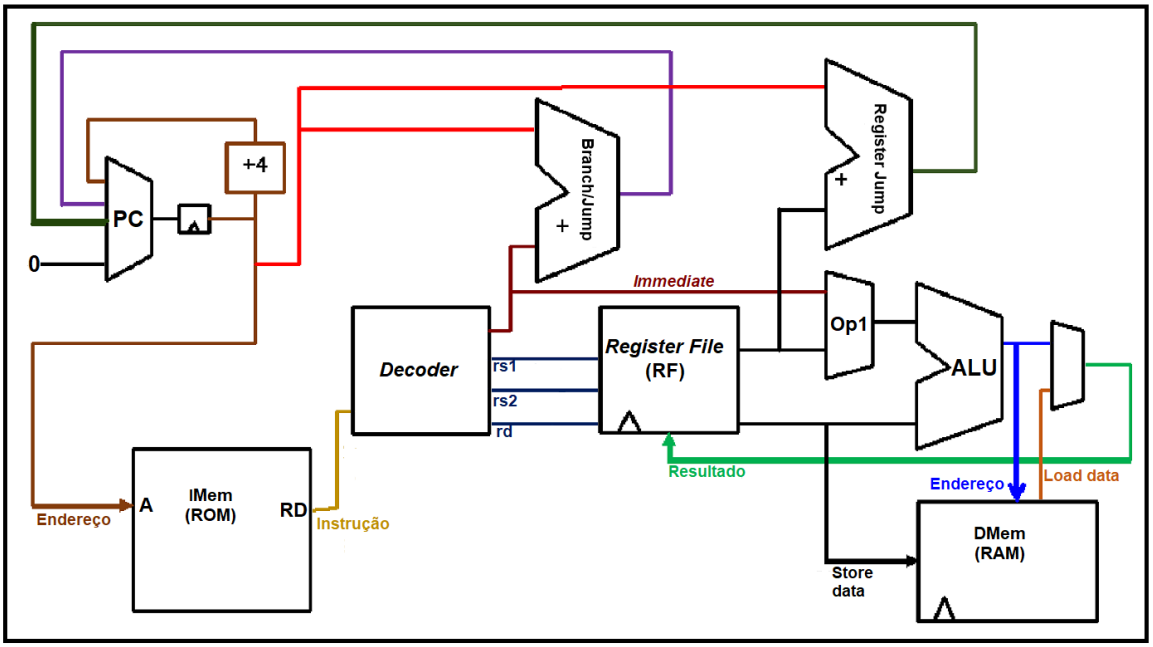
do próprio governo e, com isso, reduzir a dependência de tecnologias estrangeiras nessa área estratégica (HRUSKA, 2021c) .

Por fim, o RISC-V ganhou 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 RISC-V, com a criação da IFS (*Intel® Foundry Services*), oferecendo à SiFive a produção de CPUs RISC-V utilizando o processo de 7 nm (HRUSKA, 2021b; HRUSKA, 2021a). 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 uma oferta pública inicial (*initial public offering*, IPO) para abrir o capital da empresa no mercado financeiro (DAHAD, 2022). Já a Intel®, anunciou as suas novidades em fevereiro de 2022, quando ocorreu o anúncio de um fundo de desenvolvimento bilionário, com foco no desenvolvimento de novas CPUs RISC-V, desenvolvidas para serem fabricadas nas plantas da IFS (HRUSKA , 2022a). 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 , 2022b).

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

A Figura 1 apresenta o diagrama esquemático básico de um MCU RISC-V com o conjunto de instruções RV32IM, desenvolvido aqui. O MCU possui 33 registradores, sendo 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). Esse conjunto de registradores se chama *register file* (RF) e são as memórias mais rápidas em qualquer CPU (NISSAM e SCHOCKEN, 2021; HARRIS e HARRIS, 2022). No caso do MCU criado aqui, o registrador `pc` e os 32 registradores do RF possuem 32 bits de comprimento.

Figura 1 – Diagrama esquemático do núcleo RV32IM desenvolvido aqui.



Fonte: Material adaptado do curso on-line "*Building a RISC-V CPU Core*" (Hoover, 2021).

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 de apenas leitura (*Read-Only Memory*, ROM) – bloco "IMem" na Figura 1 – para armazenar o programa em execução (*program memory*), e uma memória de acesso aleatório (*Random-Access Memory*, RAM) – 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 grupos apresentados nessa tabela (WATERMAN, 2016). Já o subconjunto RV32M contém mais 8 instruções que seguem o formato *R-Type* apresentado na primeira linha da Tabela 2 (PATTERSON e WATERMAN, 2017).

Quanto às instruções decodificadas no *Decoder*, é importante ressaltar que todas as instruções armazenadas em IMem possuem, invariavelmente, 32 bits de comprimento (RISC-V INTERNATIONAL, 2022).

Outro componente vital do MCU (Figura 1) é o bloco ALU. A ALU (*Arithmetic Logic Unit*, ALU) é o componente responsável por realizar as operações lógicas e aritméticas do MCU desenvolvido aqui. Esse componente 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 nas entradas da 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.

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).

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.

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).

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

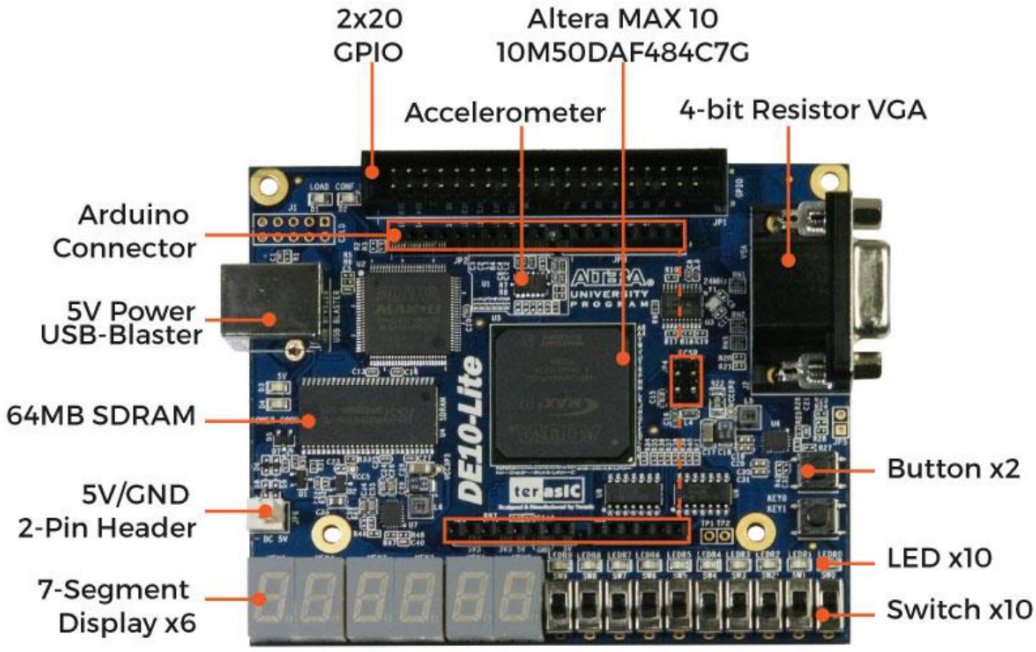
Fonte: RISC-V INTERNATIONAL (2022).

A próxima seção apresenta o desenvolvimento do núcleo RV32IM apresentado aqui.

3 METODOLOGIA

O núcleo desenvolvido aqui foi descrito utilizando as linguagens de descrição de hardware (HDLs) Verilog e SystemVerilog. Ele foi gravado e testado em um kit Terasic® DE10-Lite (Figura 2), que é baseado em um circuito integrado (CI) com arranjo de portas programáveis em campo (*Field-Programmable Gate Array*, FPGA) da família MAX 10 (modelo 10M50DAF484C7G) da Altera® (atualmente Intel® Corporation). A ferramenta de desenvolvimento integrada (IDE) utilizada foi o programa *Intel® Quartus® Prime 20.1.1 Lite Edition*. As subseções a seguir detalham os recursos e o desenvolvimento realizado.

Figura 2 – Kit FPGA Terasic DE10-Lite.



Fonte: Manual do kit Terasic DE10-Lite.

3.1 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 (*logic elements*, LEs), 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® Prime*.

A **Tabela 4** apresenta um resumo dos recursos de hardware utilizados para o desenvolvimento da CPU criada aqui.

Tabela 4 – Recursos utilizados do CI FPGA 10M50DAF484C7G.

	Total	Usados	% Utilização
Elementos Lógicos (LEs)	49.760	5.775	11,6
Pinos	360	51	14,2
Memory Bits	1.677.312	32.736	2,0
Multiplicadores de 9 bits	288	16	5,6

Fonte: *Compilation Report* do projeto `Proj_RV32IM_04_De10Lite_V2` (disponível no repositório).

Através da Tabela 4 é possível concluir que a CPU desenvolvida consome apenas uma fração dos recursos disponíveis no FPGA utilizado. Sendo que os elementos lógicos, que representam os recursos fundamentais para qualquer hardware sintetizado, não chegaram a consumir 12 % da quantidade disponível.

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

3.2 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 Institute of Electrical and Electronics Engineers (IEEE), com o pronunciamento 1364-1995 e, posteriormente, atualizada em 2001 e em 2005 (IEEE Std. 1364-2001[6]).

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[7]. 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.

3.3 Instruções implementadas

A Tabela 5 apresenta o formato das instruções do conjunto RV32IM que foram implementadas no núcleo de processamento desenvolvido.

[6] Disponível em: https://ieeexplore.ieee.org/document/1620780
[7] Disponível em: https://ieeexplore.ieee.org/document/5354441

Tabela 5 – Instruções de *Assembly* RISC-V do conjunto de instruções RV32IM.

Tipo	Subgrupo	Formato
R-Type	**Aritmética**	<instr> rd, rs1, rs2
	Lógica	<instr> rd, rs1, rs2
	Deslocamento	<instr> rd, rs1, rs2
	Menor ou Igual	<instr> rd, rs1, rs2
S-Type	***Store***	<instr> rs1, imm(rs2)
B-Type	***Branch***	<instr> rd, rs1, imm
U-Type	***Load Data***	<instr> rd, rs1
J-Type	***Jump-and-Link***	<instr> rd, imm
I-Type	***Jump-and-link Reg***	<instr> rd, imm(rs1)
	Aritmética	<instr> rd, rs1, imm
	Lógica	<instr> rd, rs1, imm
	Deslocamento	<instr> rd, rs1, imm
	Menor ou Igual	<instr> rd, rs1, imm
	Load Data	<instr> rd, rs1, imm

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

Onde na tabela está escrito <instr> deve ser substituído para mnemônico correspondente a operação a ser realizada. Por exemplo, uma operação aritmética de adição deve ser completada com o mnemônico add. Sendo rd o registrador de destino do resultado e rs1 e rs2 os registradores de origem.

3.4 Repositório com os arquivos do projeto

Com o objetivo de facilitar o acesso a todo material criado e utilizado para teste, foi criado um repositório no Github com todos os projetos usados nos testes desse trabalho[8]. 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.

O primeiro projeto do repositório, pasta `Proj_RV32I_01_ALU_RISC-V`, realiza a depuração manual da ALU. Os códigos das instruções são selecionados por meio das chaves do kit DE10-Lite e o a 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.

O segundo projeto na pasta `Proj_RV32IM_02_De10Lite` contém uma subversão do projeto final, onde o módulo principal é testado através do banco de testes descrito no arquivo `testbench.sv` por meio da ferramenta *ModelSim®*, que acompanha o software *Quartus® Prime.*

Por fim, o projeto final está na pasta `Proj_RV32IM_03_De10Lite`, nesta encontram-se os arquivos de descrição do MCU e *scripts* em *Assembly* RISC-V, que foram convertidos em códigos de máquina para verificação se as instruções são executadas corretamente pela CPU. Esses scripts estão disponíveis no Quadro 1 e no Quadro 2.

Os programas já convertidos para código de máquina foram enviados para a *instruction memory* (memória ROM) da CPU e os resultados das instruções executadas pelos programas foram armazenados na *data memory* (memória RAM). O conteúdo da memória RAM foi

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

analisado através da ferramenta *In-System Memory Content Editor*. Esse passo será mostrado nas próximas seções.

3.5 Programa em *Assembly* RISC-V com o primeiro teste

O **Quadro 1** apresenta um dos programas em *Assembly*, presente na pasta `Proj_RV32IM_03_De10Lite`, que realiza a soma de 0 a 9, armazena o resultado no registrador `x2`, que na sequência tem seu valor copiado para o registrador `x6`, do que é subtraído o valor 44 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)   # A - MEM[4 + 0] = 4 (0x04)
lw  x10, 0(x1)
sw   x2, 4(x1)   # B - MEM[4 + 4] = 45 (0x2D)
lw  x10, 4(x1)
sw   x3, 8(x1)   # C - MEM[4 + 8] = 10 (0x0A)
lw  x10, 8(x1)
sw   x4, 12(x1)  # D - MEM[4 + 12] = 10 (0x0A)
lw  x10, 12(x1)
sw  x5, 16(x1)   # E - MEM[4 + 16] = 0 (0x00)
lw  x10, 16(x1)

sw   x7, 20(x1)  # F - MEM[4 + 20] = 45 (0x2D)
lw  x10, 20(x1)
sw   x6, 24(x1)  # G - MEM[4 + 24] = 1 (0x01)
lw  x10, 24(x1)
# Bloco 'end'
end: beq  x0, x0, end     # Encerra o programa
```

Fonte: Hoover (2021).

Quadro 2 – *Script* de teste 2: Instruções diversas.

```
# Registradores com os valores usados nas operacoes
addi x12, x0, 23
addi x13, x0, 3
# Salvar os valores dos registradores
sw   x12, 4(x0)    # A – x12 = 23 (0x17)
lw   x3,  4(x0)
sw   x13, 8(x0)    # B – x13 = 3 (0x03)
lw   x3,  8(x0)
# 'add'
add x2, x12, x13   # x2 = 26 (0x1A)
sw  x2, 12(x0)     # C - MEM[16] = 26 (0x1A)
lw  x3, 12(x0)
# 'sub'
sub x2, x12, x13   # x2 = 20 (0x14)
sw  x2, 16(x0)     # D - MEM[20] = 20 (0x14)
lw  x3, 16(x0)
# 'and' / 'andi'
and x2, x12, x13   # x2 = 3 (0x03)
sw  x2, 20(x0)     # E - MEM[24] = 3 (0x03)
lw  x3, 20(x0)
# 'or' / 'ori'
or x2, x12, x13    # x2 = 23 (0x17)
sw x2, 24(x0)      # F - MEM[28] = 23 (0x17)
lw x3, 24(x0)
# 'xor' / 'xori'
xor x2, x12, x13   # x2 = 20 (0x14)
sw  x2, 28(x0)     # G - MEM[32] = 20 (0x14)
lw  x3, 28(x0)
# 'sll' / 'slli'
sll x2, x12, x13   # x2 = 184 (0xB8)
sw  x2, 32(x0)     # H - MEM[36] = 184 (0xB8)
lw  x3, 32(x0)
# 'srl' / 'slti'
srl x2, x12, x13   # x2 = 2 (0x02)
sw x2, 36(x0)      # I - MEM[40] = 2 (0x02)
lw x3, 36(x0)
# 'slt' / 'slti'
slt x2, x12, x13   # x2 = 0 (0x00)
sw x2, 40(x0)      # J - MEM[44] = 0 (0x00)
lw x3, 40(x0)
# 'sra' / 'srai'
sra x2, x12, x13   # x2 = 2 (0x02)
sw  x2, 44(x0)     # K - MEM[48] = 2 (0x02)
lw  x3, 44(x0)
# 'mul'
mul x2, x12, x13   # x2 = (x12  * x13)[31:0] = 69 (0x45)
sw  x2, 48(x0)     # L - MEM[52] = 69 (0x45)
lw  x3, 48(x0)
# 'mulh'
mulh x2, x12, x13  # x2 = (x12  * x13)[63:32] = 0 (0x00)
sw   x2, 52(x0)    # M - MEM[56] = 0 (0x00)
lw   x3, 52(x0)
# 'div'
div x2, x12, x13  # x2 = x12 / x13 = 7 (0x07)
sw  x2, 56(x0)    # N - MEM[60] = 7 (0x07)
lw  x3, 56(x0)
# 'rem'
rem x2, x12, x13  # x2 = x12 % x13 = 2 (0x02)
sw  x2, 60(x0)    # O - MEM[64] = 2 (0x02)
lw  x3, 60(x0)
# Bloco 'end'
end: beq x0, x0, end     # Encerra o programa
```

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

Nos comentários do Quadro 1, as letras de A a G apresentam os valores dos resultados esperados, as mesmas letras são apresentadas na Figura 4 para indicar a posição na memória onde tais resultados foram armazenados e permitir a verificação de sucesso nos testes.

3.6 Programa em Assembly RISC-V com o segundo teste

O Quadro 2 apresenta o segundo programa em Assembly, presente na pasta Proj_RV32IM_03_De10Lite, 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.

Nos comentários do Quadro 2, as letras de A a O apresentam os valores dos resultados esperados, as mesmas letras são apresentadas na Figura 5 para indicar a posição na memória onde tais resultados foram armazenados e permitir a verificação de sucesso nos testes.

4 RESULTADOS E DISCUSSÕES

A primeira subseção apresenta os testes do projeto presente na pasta `Proj_RV32IM_02_De10Lite` no repositório compartilhado no Github. A segunda e a terceira subseções apresentam os testes no projeto presenta na pasta `Proj_RV32IM_03_De10Lite`. A última subseção apresenta os diagramas esquemáticos do hardware gerados pela ferramenta *Quartus® Prime Lite Edition* 20.1.1.

4.1 Projeto `Proj_RV32IM_02_De10Lite`, resultado em formas de ondas (*waveforms*) da simulação

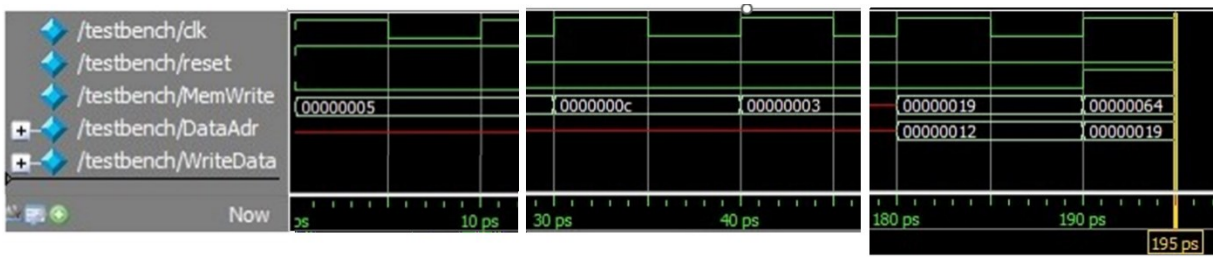
O projeto da pasta `Proj_RV32IM_02_De10Lite` é uma versão ampliada do projeto de CPU RV32I *Single-Cycle* apresentado em Harris e Harris (2022), cujo arquivo Assembly não foi apresentado nos quadros anteriores. O resultado de sua simulação é apresentado em formas de ondas exibidas na Figura 3.

A Figura 3 ilustra parcialmente os resultados da simulação do comportamento do programa testado no CI FPGA. Os sinais utilizados na simulação são 1) `clk` (Sinais do *clock*), 2) `reset` (reset assíncrono), 3) `MemWrite` (sinal indicando que está sendo realizada uma operação de escrita), 4) `DataAdr` (endereço de memória onde será realizada uma operação de leitura ou de escrita) e 5) `WriteData` (conteúdo a ser escrito na memória). Conforme já descrito, o MCU simulado é do tipo *single-cycle*, de forma que cada uma das instruções do programa é executada no período de um ciclo de *clock* (HARRIS e HARRIS, 2022). É possível observar que a CPU executa as instruções sempre nas bordas de subida do *clock* (*positive-edge triggered*). Logo, os valores dos sinais `DataAdr` e `WriteData` são atualizados sempre quando o sinal de *clock* é igual a 1.

A

Tabela **6** apresenta o resultado de todas as instruções executadas no início de cada ciclo de *clock* e os respectivos valores dos sinais `DataAdr` e `WriteData` após a execução. Na Figura 3 são destacados apenas cinco das dezoito instruções executadas para facilitar a visualização. Todos os valores apresentados estão em base hexadecimal e representam os instantes 0, 30, 40, 180 e 190 ps. Os resultados apresentados parcialmente na figura e na tabela são iguais aos resultados obtidos na simulação do exemplo original e, com isso, indicam que o código do microcontrolador implementado aqui está correto e foi executado de acordo com o esperado. Com os resultados confirmados, a próxima etapa foi testar os programas em Assembly RISC-V criados para esse trabalho.

Figura 3 – Forma de onda (*waveform*) da simulação.



Fonte: Adaptado da saída gerada pelo ModelSim®.

Tabela 6 – Instruções que geraram os resultados apresentados na Figura 2.

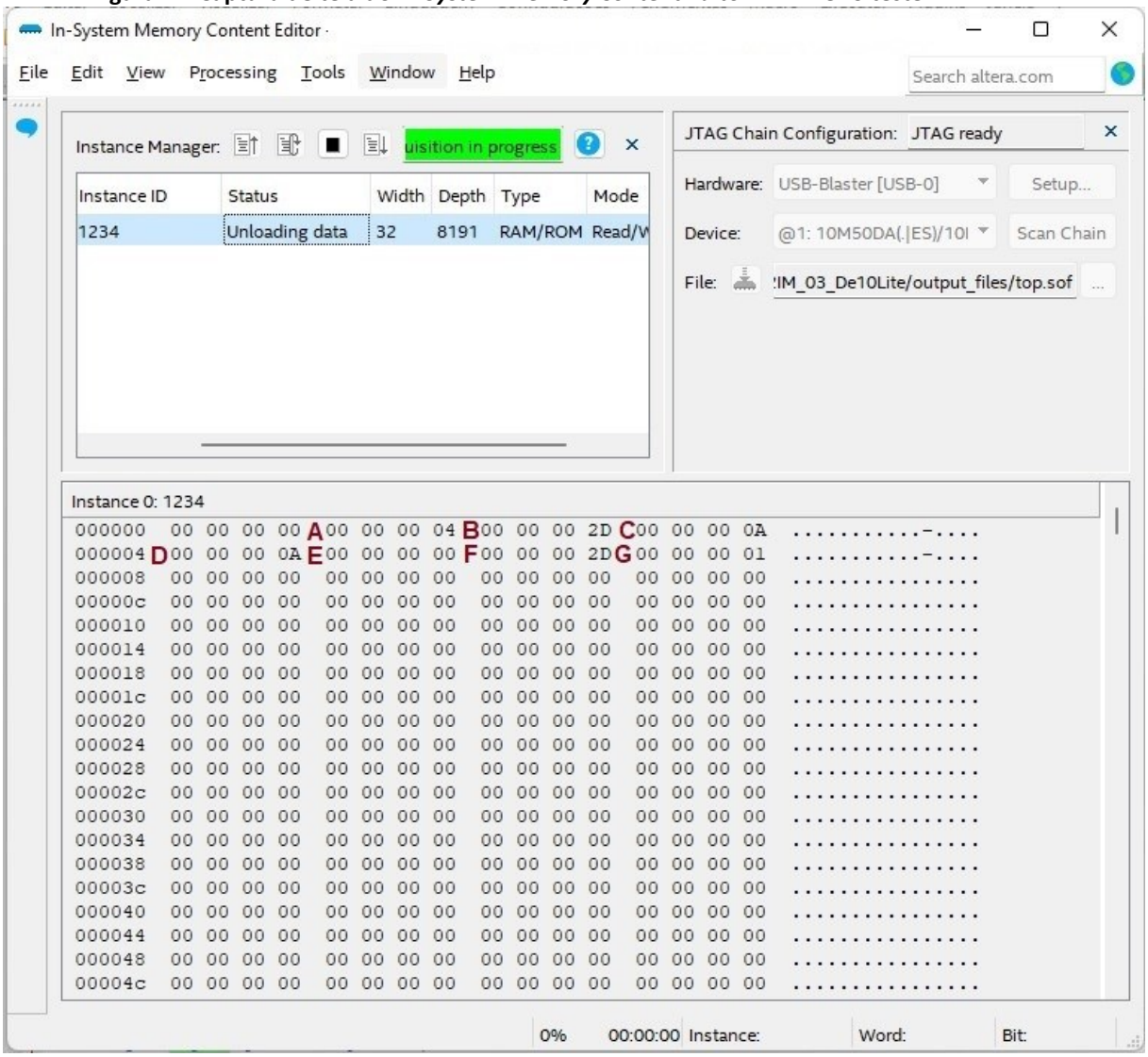
t	Instrução	DataAdr	WriteData	Sobre a Instrução
0 ps	main: addi x2, x0, 5	0x00000005	X	x2 = 0 + 5 = 5
30 ps	addi x3, x0, 12	0x0000000C	X	x3 = 0 + 12 = 12
40 ps	addi x7, x3, -9	0x00000003	X	x7 = x3 - 9 = 12 - 9 = 3
50 ps	or x4, x7, x2	0x00000007	0x00000005	x4 = x3 | x7 = 3 | 5 = 7
60 ps	and x5, x3, x4	0x00000004	0x00000007	x5 = x3 & x4 = 12 & 7 = 4
70 ps	add x5, x5, x4	0x0000000B	0x00000007	x5 = x5 + x4 = 4 + 7 = 11
80 ps	beq x5, x7, end	0x00000008	0x00000003	Se 11 < 7, ir para "end"
90 ps	slt x4, x3, x4	0x00000000	0x00000007	x4 = x3 < x4 = 12 < 7 = 0
100 ps	beq x4, x0, around	0x00000000	0x00000000	Se x4 == 0, ir para "around"
110 ps	around: slt x4, x7, x2	0x00000001	0x00000005	x4 = x7 < x2 = 3 < 5 = 1
120 ps	add x7, x4, x5	0x0000000C	0x0000000B	x7 = x4 + x5 = 1 + 11 = 12
130 ps	sub x7, x7, x2	0x00000007	0x00000005	x7 = x7 - x2 = 12 - 5 = 7
140 ps	sw x7, 84(x3)	0x00000060	0x00000007	DMem[84 + 12] = DMem[96] = 7
150 ps	lw x2, 96(x0)	0x00000060	X	x2 = DMem[96 + 0] = DMem[96]
160 ps	add x9, x2, x5	0x00000012	0x0000000B	x9 = x2 + x5 = 7 + 11 = 18
170 ps	jal x3, end	X	X	Ir para "end"; x3 = IMem[end]
180 ps	end: add x2, x2, x9	0x00000019	0x00000012	x2 = x2 + x9 = 7 + 18 = 25
190 ps	sw x2, 0x20(x3)	0x00000064	0x00000019	DMem[32 + x3] = DMem[100] = 25

Fonte: Elaborada pelos autores.

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. No Quadro 1 foram indicados os valores esperados em cada endereço da memória de dados (memória RAM) ao final da execução. Destaque para o valor da soma, `45` (`0x2D`), que foi salvo nos endereços de memória `0x08` e `0x20`. Na Figura 4 é possível confirmar esses resultados, onde cada letra nessa figura corresponde ao resultado indicado na respectiva instrução do Quadro 1.

Figura 4 – Captura de tela do *In-System Memory Content Editor* – Primeiro teste.



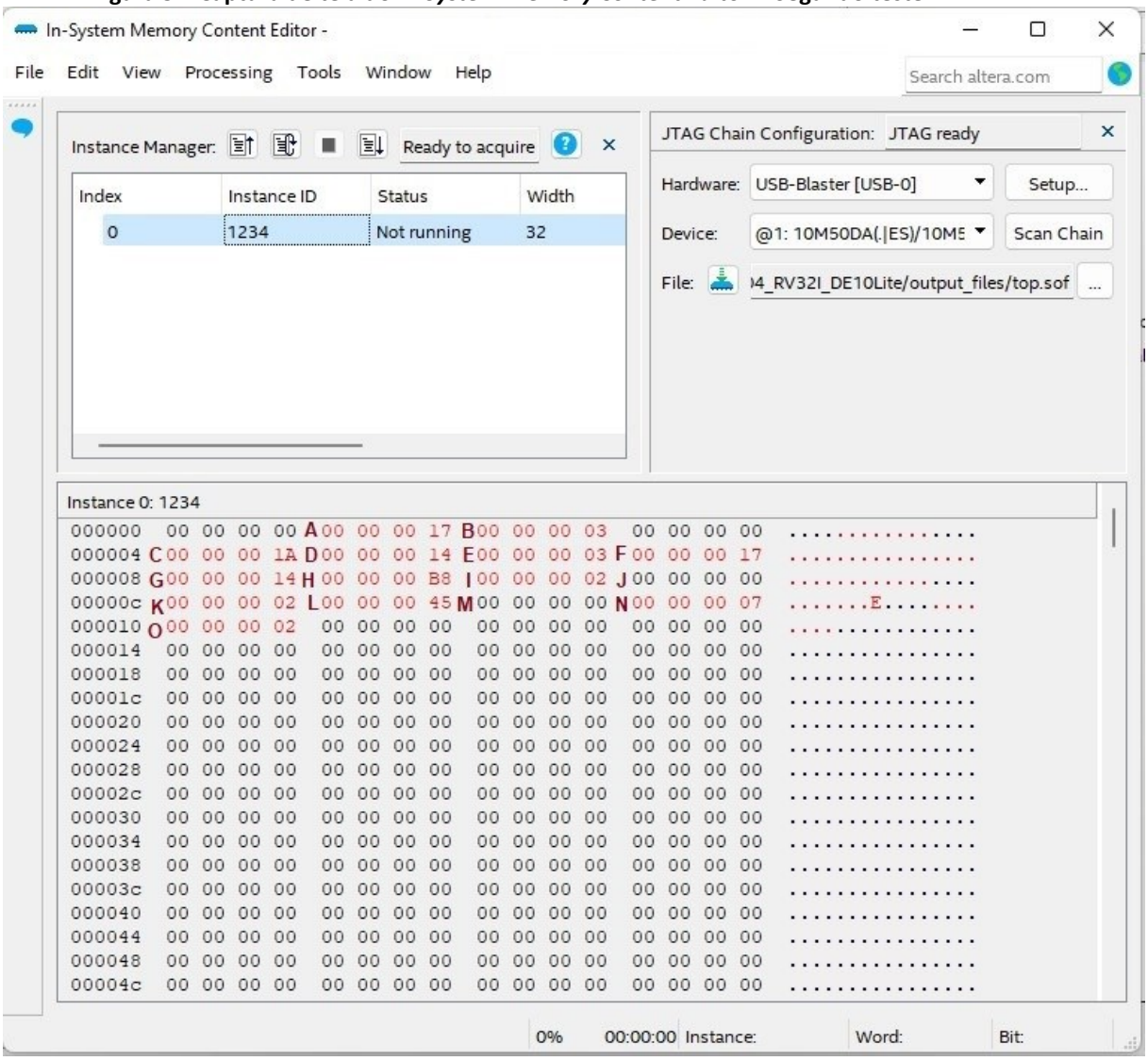
Fonte: Captura de tela do Intel® *Quartus® Prime 20.1.1 Lite Edition*.

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 execução do segundo programa, apresentado no Quadro 2, estão apresentados na Figura 5. Assim como no exemplo da subseção anterior, os valores esperados indicados nas letras dos comentários das instruções do Quadro 2 foram exatamente os mesmos que foram salvos na memória RAM da CPU desenvolvida aqui.

Figura 5 – Captura de tela do *In-System Memory Content Editor* – Segundo teste.



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

Os resultados obtidos demonstram que o MCU desenvolvido aqui é capaz de executar as instruções do conjunto base da ISA RISC-V (RV32I) e, também, as instruções de multiplicação, divisão e resto do subconjunto RV32M. Estas instruções são usadas exclusivamente com números inteiros.

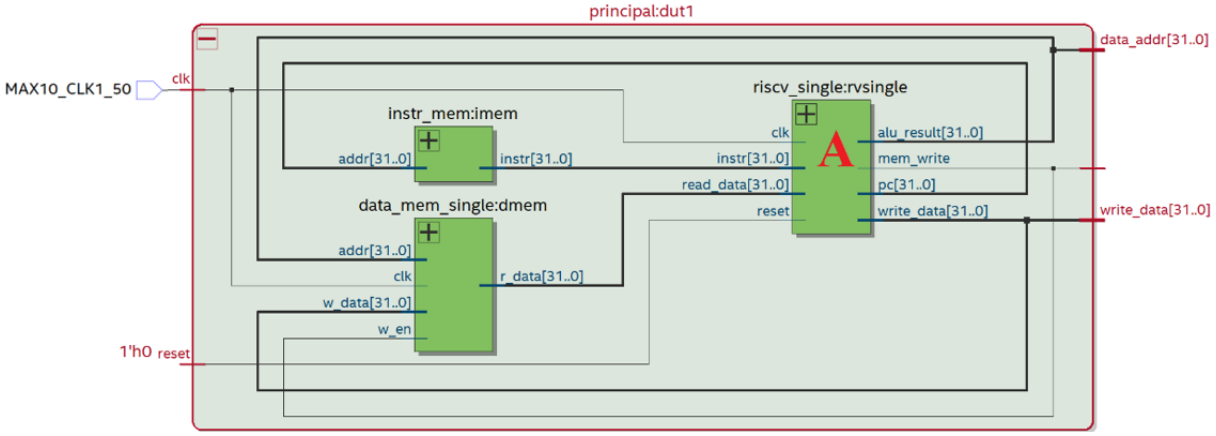
4.4 Diagramas esquemáticos do projeto

As figuras Figura **6** e Figura **7** apresentam os diagramas esquemáticos da CPU desenvolvida e o detalhamento de alguns dos seus principais blocos. Essas figuras foram geradas por veio da ferramenta *RTL Viewer* do *Quartus® Prime Lite Edition*. Apesar das figuras não serem capazes de representarem os circuitos em detalhes, elas expressam os principais módulos e suas conexões. Uma visualização mais detalhada somente é possível através da ferramenta citada.

A Figura 6 apresenta o diagrama geral do núcleo criado, que é formado pela memória ROM para armazenamento de instruções (*instr_mem*), a memória RAM para armazenamento

de dados (*data_mem*) e o núcleo RV32IM (*riscv_single,* bloco A).

Figura 6 – Estrutura básica: *program memory*, *data memory* e Núcleo RV32IM.



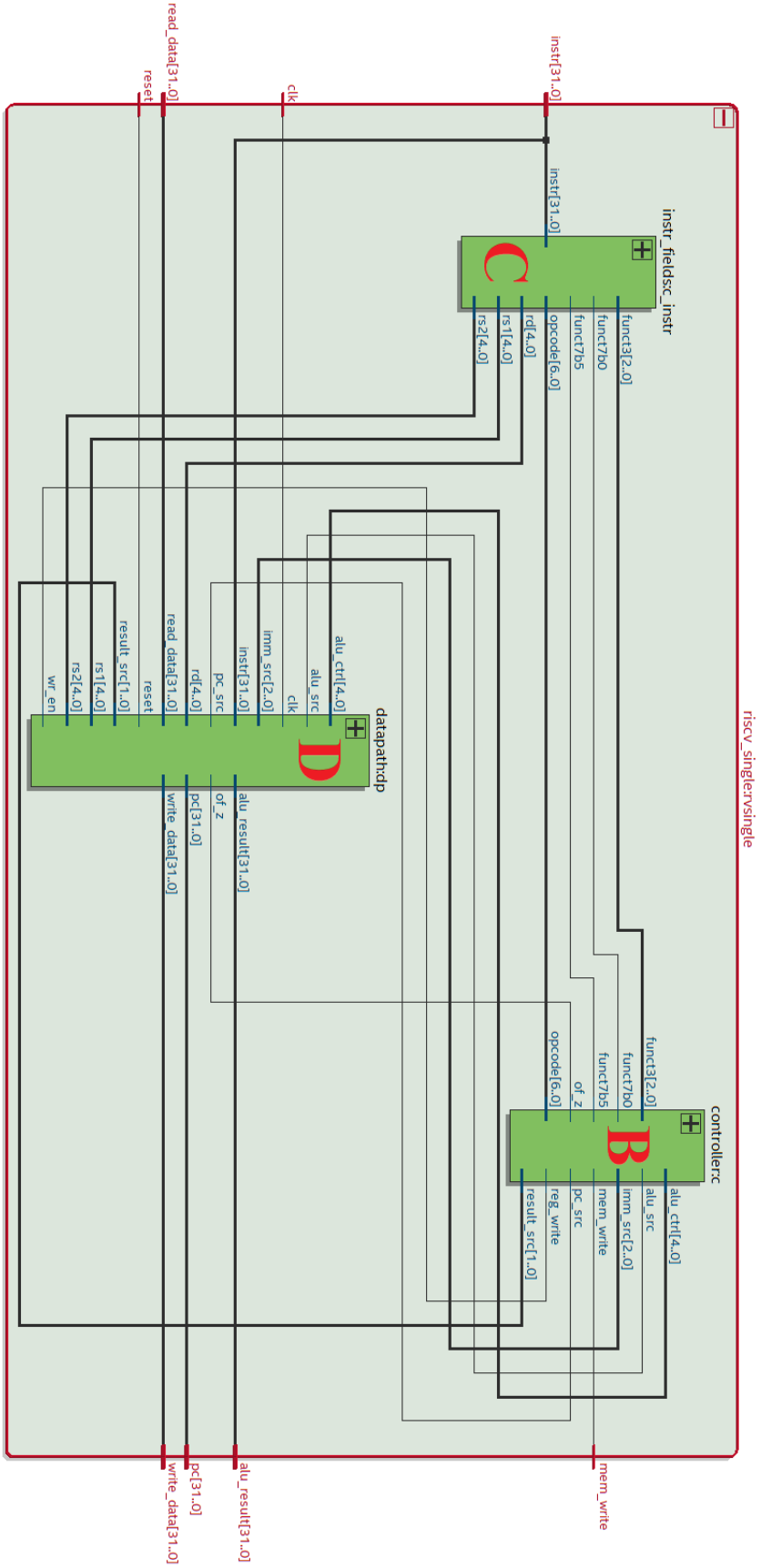
Fonte: Diagrama gerado no "*RTL Viewer*" do Intel® *Quartus® Prime 20.1.1 Lite Edition*.

O bloco A da Figura 6 é apresentado na Figura 7 expandido e destacando seus três componentes:

- *instr_field* (bloco C): é o registrador responsável por armazenar a próxima instrução ser executada. Essa Instrução é proveniente de memória ROM;
- *controller* (bloco B): recebe uma instrução do *instr_field*, decodifica a instrução recebida e, então, instrui o *datapath* (bloco D) sobre como a instrução deve ser executada (HARRIS e HARRIS, 2022);
- *datapath* (bloco D): é responsável por executar a instrução decodificada (HARRIS e HARRIS, 2022). Este bloco contém componentes vitais da CPU, como o *register file* (conjunto de registradores) e a unidade lógica e aritmética (ALU), onde são realizadas operações aritméticas e lógicas.

Dando continuidade ao trabalho, a próxima seção encerra com a apresentação das conclusões.

Figura 7 – Núcleo RV32IM (Bloco “A” da Figura 6).



Fonte: Diagrama gerado no “*RTL Viewer*” do Intel® *Quartus® Prime 20.1.1 Lite Edition.*

5 CONCLUSÃO

Este trabalho apresentou o desenvolvimento de um MCU (microcontrolador) de 32 bits que implementa o conjunto de instruções básico da ISA RISC-V, mais as operações de divisão e multiplicação. 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 desenvolvimento, a implementação e a validação do MCU desenvolvido aqui foram realizados utilizando o kit FPGA Terasic® DE10-Lite. Esse kit utiliza um CI FPGA Altera®/Intel® da família MAX 10®, cujos produtos são mais básicos (com menos recursos), mas mais baratos quando comparados com as outras famílias de CIs FPGA (TERASIC, 2020). Como mostrado na **Tabela 4**, a utilização de um hardware básico não gerou qualquer limitação ao desenvolvimento do núcleo desenvolvido. A baixa utilização de recursos corrobora com o que foi previsto por Patterson e Waterman (2017), em que uma arquitetura mais enxuta permite um melhor aproveitamento dos recursos disponíveis e um menor custo unitário dos chips produzidos.

Os testes realizados com *scripts,* especialmente desenvolvidos para validação de todo conjunto de instruções disponíveis nesta versão da MCU RISC-V, confirmaram 100% de sucesso na interpretação, execução e armazenamento dos resultados das instruções executadas. Isso classifica o dispositivo como apto a executar as instruções do conjunto RV32IM.

Após essa breve síntese, destacamos a seguir as principais contribuições deste trabalho:

1) Implementação de MCU funcional de acordo com ISA RV32IM;
2) Apresentação de metodologia e *scripts* para validação de conjunto de instruções;
3) Criação de opção de núcleo enxuto e robusto de 32 bits facilmente sintetizável em FPGAs com baixo volume de elementos lógicos;
4) Formação de recursos humanos aptos ao desenvolvimento de projetos complexos associados a arquitetura de processadores e hardware configurável. Área essa tão sensível e importante para o desenvolvimento tecnológico de qualquer nação.

Uma vez que a MCU proposta é formada apenas pelos blocos que compõe a CPU e pelas memórias, enxergamos uma séria de trabalhos futuros que podem se somar a esse e tornar essa MCU apta a ser usada em projetos de sistemas embarcados com núcleos do tipo *soft-core* sintetizados em FPGA. Seguem sugestões para trabalhos futuros:

1) Desenvolvimento de versões mais robustas e eficientes da CPU, como a inclusão de *pipelines*. Ao fracionar a execução de instruções seria possível aumentar o *clock* máximo do núcelo;

2) Inclusão de periféricos à CPU, de forma a tornar a MCU proposta completamente funcional. Exemplo: GPIOs (*general purpose input/output*) e portas de comunicação serial. Essa integração permitiria à MCU interagir com dispositivos externos, como sensores e atuadores, além de estabelecer a comunicação para troca de dados com outros dispositivos ou sistemas;

3) Validação de compiladores C para uso em conjunto com a MCU. Isso simplificaria a programação e a facilitaria a elaboração de algoritmos.

REFERÊNCIAS

ALLWINNER. Allwinner launches the first RISC-V application processor. **EE Times**, 15 abr. 2021. Disponível em: https://www.eetimes.com/allwinner-launches-the-first-risc-v-application-processor/. Acesso em: 19 jun. 2023.

ASANOVIć, Krste; PATTERSON, David A. **Instruction sets should be free: the case for RISC-V**. UC Berkeley. Berkeley, p. 7. 2014.

BAILEY, Brian. Why RISC-V is succeeding. **Semiconductor Engineering**, 24 abr. 2022. Disponível em: https://semiengineering.com/why-risc-v-is-succeeding/. Acesso em: 19 jun. 2023.

BAINES, Rupert. Differentiation and architecture licenses in RISC-V. **Semiconductor Engineering**, 26 maio 2022. Disponível em: https://semiengineering.com/differentiation-and-architecture-licenses-in-risc%e2%80%91v/. Acesso em: 19 jun. 2023.

CHU, Pong P. **Embedded SOPC design with NIOS II processor and Verilog examples**. New Jersey: John Wiley & Sons, 2012.

CORDING, Stuart. What Is RISC-V? An in-depth introduction to the RISC-V instruction set architecture. Elektor, 2021. Disponível em: https://www.elektormagazine.com/articles/what-is-risc-v. Acesso em: 19 jun. 2023.

DAHAD, Nitin. SiFive raises $175M to quicken ‘Arm intercept’ strategy. **EE Times**, 16 mar. 2022. Disponível em: https://www.eetimes.com/sifive-raises-175m-to-quicken-arm-intercept-strategy/. Acesso em: 19 jun. 2023.

EE TIMES. Why RISC-V lags in China. **EE Times**, 15 nov. 2018. Disponível em: https://www.eetimes.com/why-risc-v-lags-in-china/. Acesso em: 19 jun. 2023.

ENGHEIM, Erik. What does RISC and CISC mean in 2020? **Medium**, 27 jul. 2020a. Disponível em: https://medium.com/swlh/what-does-risc-and-cisc-mean-in-2020-7b4d42c9a9de. Acesso em: 19 jun. 2023.

ENGHEIM, Erik. What is innovative about RISC-V? **Medium**, 24 dez. 2020b. Disponível em: https://medium.com/swlh/what-is-innovative-about-risc-v-a821036a1568. Acesso em: 19 jun. 2023.

ENGHEIM, Erik. RISC-V is the king of heterogenous computing. **Medium**, 06 jan. 2022a. Disponível em: https://erik-engheim.medium.com/risc-v-the-king-of-heterogenous-computing-11b47649e691. Acesso em: 19 jun. 2023.

ENGHEIM, Erik. The case for RISC-V on desktops and servers. **Medium**, 06 jan. 2022b. Disponível em: https://erik-engheim.medium.com/the-case-for-risc-v-on-desktops-and-severs-60b0106c636b. Acesso em: 19 jun. 2023.

HARRIS, Sarah L.; HARRIS, David M. **Digital design and computer architecture - RISC-V edition**. 2022.

HOOVER, Steve. Building a RISC-V CPU core. **Linux Foundation/edX**, 2021. Disponível em: https://www.edx.org/course/building-a-risc-v-cpu-core. Acesso em: 19 jun. 2023.

HRUSKA , Joel. Intel announces billion-dollar development fund, boosts RISC-V processors. **ExtremeTech**, 09 fev. 2022a. Disponível em: https://www.extremetech.com/extreme/331461-intel-announces-billion-dollar-development-fund-boosts-risc-v-processors. Acesso em: 19 jun. 2023.

HRUSKA , Joel. Intel plans to license hybrid chips that combine ARM, RISC-V, and x86. **ExtremeTech**, 2022b. Disponível em: https://www.extremetech.com/computing/331740-intel-plans-to-license-cores-that-combine-arm-risc-v-and-x86. Acesso em: 19 jun. 2023.

HRUSKA, Joel. ARM kills its RISC-V FUD website after staff revolt. **ExtremeTech**, 12 ago. 2018. Disponível em: https://www.extremetech.com/computing/273236-arm-kills-its-risc-v-fud-website-after-staff-revolt. Acesso em: 19 jun. 2023.

HRUSKA, Joel. Cut off from ARM, x86, what CPU architectures can Huawei use? **ExtremeTech**, 23 maio 2019a. Disponível em: https://www.extremetech.com/computing/291875-cut-off-from-arm-x86-what-cpu-architectures-can-huawei-actually-use. Acesso em: 19 jun. 2023.

HRUSKA, Joel. Western digital's RISC-V 'Swerv' core now available for free. **ExtremeTech**, 15 fev. 2019b. Disponível em: https://www.extremetech.com/computing/285856-western-digitals-risc-v-swerv-core-now-available-for-free. Acesso em: 19 jun. 2023.

HRUSKA, Joel. Intel will offer SiFive RISC-V CPUs on 7nm, plans own dev platform. **ExtremeTech**, 24 jun. 2021a. Disponível em: https://www.extremetech.com/computing/324075-intel-will-offer-sifive-risc-v-cpus-on-7nm-plans-own-dev-platform. Acesso em: 19 jun. 2023.

HRUSKA, Joel. Rumor: Intel may buy RISC-V CPU designer SiFive to fend off ARM. **ExtremeTech**, 11 jun. 2021b. Disponível em: https://www.extremetech.com/computing/323647-rumor-intel-may-buy-risc-v-cpu-designer-sifive-to-fend-off-arm. Acesso em: 19 jun. 2023.

HRUSKA, Joel. Russia to build 8-Core RISC-V CPUs for laptops, government systems. **ExtremeTech**, 15 jul. 2021c. Disponível em: https://www.extremetech.com/computing/324735-russia-to-build-8-core-risc-v-cpus-for-laptops-government-systems. Acesso em: 19 jun. 2023.

LEDIN, Jim. **Modern computer architecture and organization:** learn x86, ARM, and RISC-V architectures and the design of smartphones, PCs, and cloud servers. 2020.

LINUX FOUNDATION. Introduction to RISC-V. **edX**. 2021. Disponível em: https://www.edx.org/course/introduction-to-risc-v. Acesso em: 19 jun. 2023.

MARENA, Ted. 11 myths about the RISC-V ISA. **Electronic Design**, 31 jan. 2018. Disponível em: https://www.electronicdesign.com/technologies/embedded-revolution/article/21806096/11-myths-about-the-riscv-isa. Acesso em: 19 jun. 2023.

MERRIT, Rick. Google, HP, Oracle join RISC-V. **EE Times**, 28 dez. 2015. Disponível em: https://www.eetimes.com/google-hp-oracle-join-risc-v/. Acesso em: 19 jun. 2023.

MOORE, Samuel K. RISC-V AI chips will be everywhere. **IEEE Spectrum**, 24 fev. 2022. Disponível em: https://spectrum.ieee.org/risc-v-ai. Acesso em: 19 jun. 2023.

NISSAM, Noam; SCHOCKEN, Shimon. **The elements of computing systems**. 2a. ed.
PATTERSON, David A.; DITZEL, David R. The case for the reduced instruction set computer. **ACM SIGARCH Computer Architecture News**, 8, n. 6, 1980. 25–33.

PATTERSON, David A.; HENESSY, John L. **Computer organization and design - RISC-V edition**. 2. ed. 2021.

PATTERSON, David; WATERMAN, Andrew S. **Guia prático RISC-V:** atlas de uma arquitetura aberta. 1. ed. 2017.

RISC-V INTERNATIONAL. **The RISC-V instruction set manual - unprivileged ISA**. 2022.

SPERLING, Ed. Which processor is best? **Semiconductor Engineering**, 01 mar. 2022. Disponível em: https://semiengineering.com/which-processor-is-best/. Acesso em: 19 jun. 2023.

TERASIC. **DE10-Lite User Manual**. 2020.

TURLEY, Jim. Why universities want RISC-V. **EE Journal**, 27 out. 2020. Disponível em: https://www.eejournal.com/article/why-universities-want-risc-v/. Acesso em: 19 jun. 2023.

URQUHART. Is RISC-V the future? **Semiconductor Engineering**, 29 jul. 2021. Disponível em: https://semiengineering.com/is-risc-v-the-future/. Acesso em: 19 jun. 2023.

WATERMAN, Andrew S. **Design of the RISC-V instruction set architecture**. Berkeley: University of California, 2016.

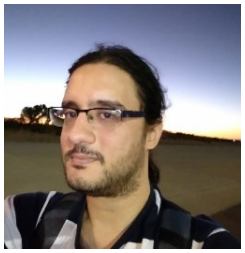
WESTERN DIGITAL. **RISC-V and open source hardware address new compute requirements**.. San Jose: Western Digital, 2019.

AGRADECIMENTOS

Agradecemos o colega Diego Salviano Nagai pelas verificações dos *scripts* e resultados desenvolvidos aqui.

SOBRE OS AUTORES

[i] **EDUARDO ALVIM GUEDES ALCOFORADO**



Economista Empresarial e Controladoria (ECEC) pela Universidade de São Paulo (2011), especialista em *Data Science* por Johns Hpkings University-Coursera (2016), Mestre em Ciências Contábeis pela Universidade Federal de Uberlândia (2017) e Especialista em Sistemas Embarcados pela Faculdade de Tecnologia SENAI (2022). http://lattes.cnpq.br/0205554239317512

[ii] **LEANDRO POLONI DANTAS**

Engenheiro (2004) e Doutor (2018) em Engenharia Elétrica pelo Centro Universitário FEI. Atuou por 15 anos na indústria eletrônica no desenvolvimento de novos produtos. Desde 2009, vem lecionando em cursos de pós-graduação, graduação e de nível técnico em diferentes instituições paulistanas. Atualmente é professor na Faculdade de Tecnologia SENAI e no Insper. https://orcid.org/0000-0003-3674-336X

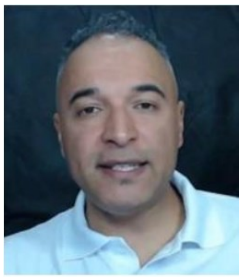
[iii] **MARCONES CLEBER BRITO DA SILVA**

Tecnólogo em Mecatrônica Industrial (2011), Engenheiro Mecatrônico (2013) e Especialista em Engenharia de Manutenção Industrial pela Centro universitário Eniac (2013). Mestre em Tecnologia Nuclear (2020) pela Universidade de São Paulo. Desde 2011, vem lecionando em cursos de nível técnicos e de graduação. Atualmente é professor da Faculdade de Tecnologia SENAI e na FESA. https://orcid.org/0000-0002-3690-1682

[iv] **LUIZ CARLOS CANNO**

Graduado em Tecnologia de Automação Industrial (2009) com Especialização em Gestão Empresarial pela Universidade Nove de Julho (2012), e Especialização em Docência na Educação Profissional e Tecnológica pelo SENAI CETIQT (2015). Professor na Faculdade de Tecnologia SENAI nos cursos graduação e pós-graduação. https://orcid.org/0000-0001-9331-9309

[v] **FERNANDO SIMPLICIO DE SOUSA**



Professor da Faculdade SENAI no curso de Pós-Graduação em Sistemas Embarcados. Mestre em Engenharia Elétrica pela Universidade Federal do ABC (UFABC) e Pós-Graduado (Lato Sensu) pela Universidade Mackenzie. Graduado em Gestão de Pequenas e Médias Empresas pela UNIP e em Projetos Mecânicos pela Faculdade de Tecnologia de São Paulo (UNESP/FATEC-SP). https://orcid.org/0009-0009-5760-4845