Estudo e desenvolvimento de uma microarquitetura RISC-V baseado no ISA RV32I.

Guilherme A. R. d. Paula Jr. (IC), Gustavo Della Colleta (PQ)
Universidade Federal de Itajubá.

Palavras-chave: RISC-V, Microarquitetura, FPGA, Arquitetura aberta.

Introdução

Nos últimos anos, a arquitetura RISC-V tem ganhado destaque como uma alternativa de grande potencial às arquiteturas de conjunto de instruções proprietárias, como ARM e x86, graças à sua abordagem aberta, modular e flexível. Desenvolvida inicialmente pela Universidade da Califórnia, Berkeley, a RISC-V oferece a pesquisadores e engenheiros a possibilidade de criar e adaptar processadores para uma vasta gama de aplicações, sem os custos de licenciamento. Como mencionado por Patterson [3] "Conjuntos de instruções abertos, como o RISC-V, permitem que organizações construam seus próprios processadores sem precisar negociar uma licença, o que possibilitou implementações de código aberto que podem ser baixadas e usadas livremente, bem como implementações proprietárias do RISC-V", o que é uma vantagem significativa em relação a ISAs proprietárias, como a ARM.

De acordo com Harris et al. Harris [1] menciona que um dos principais desafios no design de microarquiteturas é equilibrar o desempenho, o consumo de energia e a área ocupada pelo circuito. Porém a adaptação para o dispositivo FPGA confirmou seu potencial para ser utilizada em sistemas embarcados e aplicações de baixo consumo energético. O RISC-V é projetado para ser modular, permitindo a adição ou remoção de extensões conforme a necessidade, o que o torna ideal para aplicações específicas.

O presente trabalho tem como objetivo explorar a implementação de uma microarquitetura baseada na ISA RV32I da RISC-V, com foco no desenvolvimento de um processador de ciclo único (single-cycle) em FPGA, utilizando ferramentas como Quartus Prime e ModelSim. A pesquisa visa estudar a viabilidade do RISC-V em aplicações de baixo consumo energético e alto desempenho, validando sua execução de um conjunto básico de instruções em um ambiente de hardware flexível.

Ao fornecer uma plataforma prática para futuras pesquisas e implementações em sistemas de arquiteturas abertas, este estudo contribui para o entendimento das vantagens do RISC-V. A arquitetura, com sua natureza acessível e customizável, surge como uma solução promissora para projetos de hardware personalizados, evidenciando seu potencial em um mercado cada vez mais voltado para soluções específicas e eficientes.

Metodologia

O desenvolvimento da microarquitetura RISC-V baseada no conjunto de instruções RV32I foi realizado em várias etapas, abrangendo desde a análise teórica até a implementação prática em FPGA. A seguir, descreve-se a sequência de atividades realizadas, os métodos empregados e as ferramentas utilizadas ao longo do projeto.

Inicialmente, foi conduzido um estudo detalhado da arquitetura RISC-V e de seu conjunto de instruções (ISA), focando no RV32I, uma versão de 32 bits projetada para ser simples e eficiente. O estudo incluiu a análise das instruções aritméticas, lógicas, de acesso à memória e de controle de fluxo. Para guiar a implementação, foi utilizada uma microarquitetura de referência descrita na literatura especializada.

Após a compreensão da ISA, foi realizada a descrição da microarquitetura utilizando a linguagem de descrição de hardware Verilog. Essa linguagem foi escolhida pela sua capacidade de sintetizar circuitos digitais e pela compatibilidade com as ferramentas de desenvolvimento utilizadas. O código inicial foi baseado na microarquitetura de referência, sendo adaptado para ser compatível com o ambiente de desenvolvimento Quartus Prime e a plataforma FPGA DE10-Lite da Intel.

A adaptação do código envolveu ajustes específicos para otimização do uso de recursos da FPGA, garantindo o bom desempenho do processador single-core e single-cycle implementado. Testes de funcionalidade foram realizados através de simulações utilizando o software ModelSim, onde foi possível verificar a correta execução das instruções previstas pela ISA RV32I. O uso do ModelSim permitiu observar o comportamento dos sinais e identificar eventuais erros de sincronização e temporização.

Com a microarquitetura validada via simulação, foi realizada a síntese do código Verilog no Quartus Prime, onde o design foi mapeado para a FPGA DE10-Lite. A síntese incluiu a geração de arquivos binários para a programação do hardware, que, por sua vez, foi carregado

na FPGA para execução em ambiente físico. Para testar a implementação, foram utilizados programas simples em assembly RISC-V, os quais validaram a funcionalidade do processador diretamente na placa FPGA.
Durante os testes em hardware, foram identificados pequenos problemas de temporização, que foram corrigidos com ajustes nos sinais de controle e na lógica da unidade de controle. Após a correção desses problemas, foi possível validar o funcionamento completo do processador, confirmando a viabilidade do design e sua compatibilidade com a ISA RV32I.
A metodologia seguiu um fluxo iterativo de simulação, síntese e teste, garantindo a conformidade da microarquitetura com a especificação da ISA e a implementação eficiente no FPGA. Esta abordagem modular também permitiu explorar possíveis melhorias e futuras expansões da microarquitetura para inclusão de novos recursos.

Resultados e discussão

A implementação da microarquitetura RISC-V baseada no conjunto de instruções RV32I foi realizada com sucesso, permitindo a execução de operações aritméticas, lógicas, de acesso à memória e de controle de fluxo em um processador single-core e single-cycle. A estrutura modular da microarquitetura foi essencial para o funcionamento eficiente do sistema, com a Unidade de Controle, a ALU (Unidade Lógica e Aritmética), a Unidade de Registradores e a Memória de Dados trabalhando em conjunto para garantir a execução correta das instruções.

Unidade de Controle carrega a próxima instrução da Memória de Instruções, utilizando o valor armazenado no Program Counter (PC). O Program Counter é então incrementado automaticamente, apontando para a próxima instrução a ser executada no ciclo subsequente. Na etapa seguinte, ocorre a decodificação da instrução, onde a Unidade de Controle interpreta o opcode e define quais sinais de controle devem ser ativados, preparando os outros módulos, como a ALU e a Memória de Dados, para a execução da operação especificada.
Durante a execução de instruções aritméticas e lógicas, os operandos são lidos da Unidade de Registradores e enviados à ALU, que realiza a operação desejada, como soma ou subtração, retornando o resultado para ser escrito de volta nos registradores, conforme a instrução decodificada. A Figura 1 ilustra essa sequência, destacando como os dados fluem através dos diferentes módulos da microarquitetura para completar uma instrução.
No caso de instruções de acesso à memória (como load e store), o endereço de memória é calculado pela ALU, utilizando um valor imediato ou registradores, e a Memória de Dados é acessada para ler ou escrever os dados. A Unidade de Controle coordena todo o processo, garantindo que as operações ocorram de maneira sincronizada e correta.
A etapa final é o write-back, onde o resultado da operação (seja um cálculo da ALU ou um dado lido da memória) é escrito de volta em um registrador ou na memória, dependendo do tipo de instrução. Durante o ciclo, os sinais de controle garantem que os multiplexadores redirecionem os dados adequadamente entre os diferentes

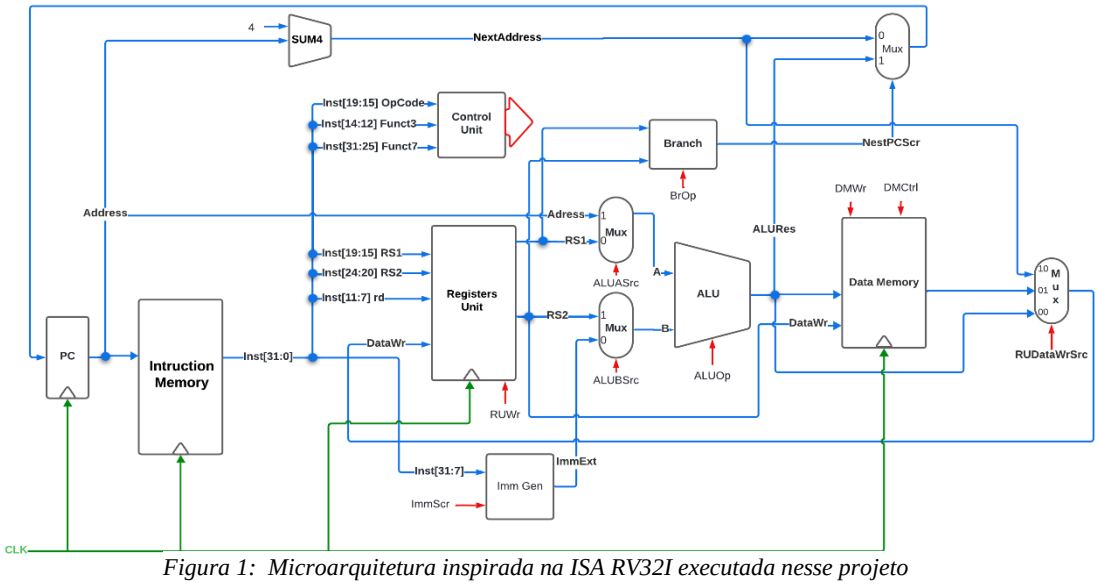

Figura 1: Microarquitetura inspirada na ISA RV32I executada nesse projeto

O processo de execução de uma instrução na microarquitetura começa com a etapa de fetch, onde a

módulos, conforme ilustrado nas figuras das simulações. Os testes de simulação realizados no ModelSim

confirmaram o comportamento correto da microarquitetura em ambiente virtual. Durante as simulações, um programa em assembly foi executado, e os resultados indicaram que as operações de armazenamento de dados na memória ocorreram conforme o esperado. A arquitetura demonstrou ser capaz de executar um ciclo completo por instrução, validando a eficiência do design.

```
# riscvtest.s
# Sarah.Harris@unlv.edu
# David_Harris@hmc.edu
# 27 Oct 2020
#
# Test the RISC-V processor:
#   add, sub, and, or, slt, addi, lw, sw, beq, jal
# If successful, it should write the value 25 to address 100
#       RISC-V Assembly         Description              Address
main:   addi x2, x0, 5          # x2 = 5                   0
        addi x3, x0, 12         # x3 = 12                  4
        addi x7, x3, -9         # x7 = (12 - 9) = 3        8
        or   x4, x7, x2         # x4 = (3 OR 5) = 7        C
        and  x5, x3, x4         # x5 = (12 AND 7) = 4      10
        add  x5, x5, x4         # x5 = 4 + 7 = 11          14
        beq  x5, x7, end        # shouldn't be taken       18
        slt  x4, x3, x4         # x4 = (12 < 7) = 0        1C
        beq  x4, x0, around     # should be taken          20
        addi x5, x0, 0          # shouldn't execute        24
around: slt  x4, x7, x2         # x4 = (3 < 5) = 1         28
        add  x7, x4, x5         # x7 = (1 + 11) = 12       2C
        sub  x7, x7, x2         # x7 = (12 - 5) = 7        30
        sw   x7, 84(x3)         # [96] = 7                 34
        lw   x2, 96(x0)         # x2 = [96] = 7            38
        add  x9, x2, x5         # x9 = (7 + 11) = 18       3C
        jal  x3, end            # jump to end, x3 = 0x44   40
        addi x2, x0, 1          # shouldn't execute        44
end:    add  x2, x2, x9         # x2 = (7 + 18) = 25       48
        sw   x2, 0x20(x3)       # [100] = 25               4C
done:   beq  x2, x2, done       # infinite loop            50
```

Figura 2: Testbench retirado da literatura para ISA RV32I

Conforme a Figura 2, o testbench simulado no ModelSim demonstrou a execução correta de instruções básicas como soma e armazenamento de dados. A simulação verificou que o número 25 foi corretamente armazenado no endereço de memória 100, indicando a precisão da execução das instruções de armazenamento e carga.
Os resultados mostraram que o processador single-cycle foi capaz de gerenciar o fluxo de controle de maneira eficaz, com a Unidade de Controle atualizando corretamente o Program Counter (PC) em cada ciclo. Pequenos ajustes de temporização foram necessários para garantir a sincronização entre os módulos, mas esses problemas foram resolvidos com modificações nos sinais de controle e multiplexadores.

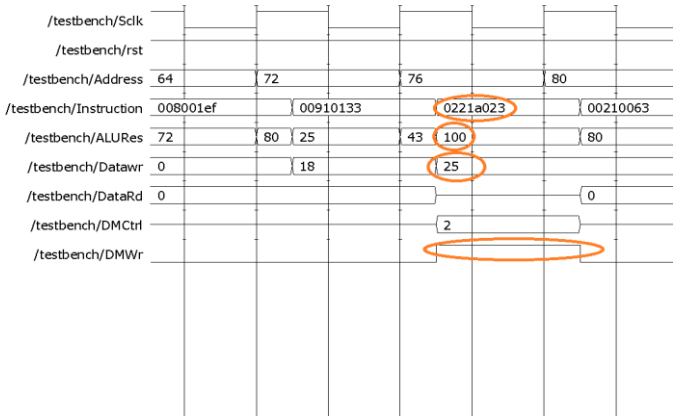

Figura 3: Resultado do testbench feito no ModelSim-Altera

Os resultados gerais da implementação validam a viabilidade da microarquitetura RISC-V em sistemas embarcados, especialmente devido à sua eficiência energética e modularidade. A implementação demonstrou que a arquitetura RISC-V é capaz de executar um ciclo por instrução, com uma organização clara entre os módulos responsáveis pelo controle, cálculos e armazenamento de dados. Além disso, os testes em FPGA demonstraram o potencial da arquitetura para futuras expansões, como a adição de novos recursos à ISA ou a implementação de um pipeline para melhorar o desempenho do processador.

Conclusões

O presente trabalho alcançou com sucesso o objetivo de desenvolver e implementar uma microarquitetura baseada no conjunto de instruções RV32I da arquitetura RISC-V, demonstrando sua viabilidade e desempenho em sistemas embarcados. A microarquitetura foi projetada de forma modular, com componentes como a ALU, a Unidade de Controle e a Memória de Dados integrados em um processador single-core e single-cycle, capaz de executar um ciclo completo por instrução.
Os testes realizados, tanto em ambiente de simulação quanto em hardware real utilizando a FPGA DE10-Lite, validaram a funcionalidade e o desempenho do design proposto. A execução correta de instruções aritméticas, lógicas, de acesso à memória e de controle de fluxo, além da solução dos desafios de temporização e sincronização, confirma a robustez e a eficiência da microarquitetura implementada.
A utilização de uma arquitetura aberta como o RISC-V destaca-se por sua flexibilidade, permitindo adaptações e expansões para atender a diferentes demandas de sistemas embarcados. Além disso, a natureza aberta e extensível dessa arquitetura oferece oportunidades para futuras pesquisas, especialmente no desenvolvimento de novos recursos, como a implementação de pipelines ou o

suporte a operações de ponto flutuante e vetorização.
Em termos de impacto, o sucesso deste projeto reforça o potencial do RISC-V como uma alternativa poderosa às arquiteturas proprietárias, sobretudo em cenários onde eficiência energética e personalização de hardware são requisitos essenciais. O projeto abre espaço para avanços futuros no campo de sistemas embarcados, arquiteturas abertas e desenvolvimento de hardware customizável.
Finalmente, este estudo destaca a importância do desenvolvimento de tecnologias baseadas em ISAs abertas como o RISC-V, não apenas pela inovação tecnológica que proporcionam, mas também por sua capacidade de democratizar o acesso ao design de processadores, permitindo que uma comunidade global contribua e evolua esse ecossistema de maneira colaborativa.

Agradecimentos

Os autores gostariam de agradecer à Universidade Federal de Itajubá (UNIFEI) pelo ambiente acadêmico e recursos fundamentais ao desenvolvimento deste projeto. Agradeço à FAPEMIG pelo apoio financeiro que viabilizou a pesquisa, permitindo meu foco integral no estudo da microarquitetura RISC-V. Também sou grato aos colegas de laboratório e amigos, por suas contribuições e apoio, e à minha família, cujo suporte emocional foi crucial para a conclusão deste trabalho.

Referências

[1] HARRIS, S. L.; HARRIS, D. Digital Design and Computer Architecture, RISC-V Edition. [s.l.] Morgan Kaufmann, 2021.

[2] HENNESSY, D. A. COMPUTER ORGANIZATION AND DESIGN RISC-V EDI TION : the hardware software interface. S.L.: Morgan Kaufmann Publisher, 2021.

[3] PATTERSON, D. A.; WATERMAN, A. The RISC-V Reader. [s.l: s.n.].

[4] CDLAVILA. GitHub- cdlavila/unicycle-processor-risc-v-rv32i: This is a SystemVe rilog implementation of the Unicycle RISC-V RV32I processor and this was the final project for Computer Architecture class. Disponível em: GitHub cdlavila. Acesso em: 12 set. 2024.

[5] TERASIC. DE10-Lite User Manual. 2016. Disponível em: Terasic.[Acesso em: 12 set. 2024].

[6] INTEL. Quartus Prime Standard Edition: Software de Design de FPGA e CPLD. Versão 20.1. 2020.