Implementação e Verificação de um Processador RISC-V Monociclo para um Subconjunto de Instruções em Verilog

Victor Cabral, Otávio Guimarães

[1]Universidade Federal de Ouro Preto (UFOP) – ICEA
João Monlevade, MG – Brasil

{victor.cabral, otavio.agf}@ufop.edu.br

Abstract. *This work presents the design, implementation, and verification of a single-cycle RISC-V processor in Verilog, supporting the instruction set* `lh`*,* `sh`*,* `sub`*,* `or`*,* `andi`*,* `srl` *and* `beq`*. Essential modules were developed and integrated, including ALU, register file, memories, and a two-level control unit. Validation was performed through a specific test program, with execution monitored and analyzed via simulation in Icarus Verilog and waveform visualization in GTKWave, confirming the correct functionality of each instruction.*

Resumo. *Este trabalho apresenta o projeto, implementação e verificação de um processador RISC-V de ciclo único em Verilog, com suporte para o conjunto de instruções* `lh`*,* `sh`*,* `sub`*,* `or`*,* `andi`*,* `srl` *e* `beq`*. Foram desenvolvidos e integrados os módulos essenciais, incluindo ULA, banco de registradores, memórias e uma unidade de controle de dois níveis. A validação do processador foi realizada através de um programa de teste específico, com a execução monitorada e analisada via simulação em Icarus Verilog e visualização de formas de onda no GTKWave, confirmando a correta funcionalidade de cada instrução.*

Repositório do projeto:
https://github.com/Vitor0302/Trabalho-Pratico-RISC-V

1. Introdução

O conjunto de instruções RISC-V é uma arquitetura aberta que vem ganhando espaço na indústria e academia devido à sua simplicidade e flexibilidade. O objetivo deste trabalho foi projetar e implementar um processador RISC-V monociclo capaz de executar um subconjunto de instruções, focando no entendimento do caminho de dados e da interação entre seus componentes.

O projeto foi realizado em Verilog e validado por meio de simulações, permitindo observar a propagação dos sinais e o comportamento do processador durante a execução de diferentes instruções.

2. Objetivos

O objetivo geral é implementar um processador RISC-V monociclo funcional, com suporte a um conjunto específico de instruções. Entre os objetivos específicos:

- Implementar os módulos básicos do processador: ULA, banco de registradores, memórias e unidades de controle;
- Executar corretamente as instruções: `lh`, `sh`, `sub`, `or`, `andi`, `srl`, `beq`;
- Validar a implementação através de simulações com um programa de teste;
- Analisar o comportamento do processador no GTKWave.

3. Metodologia

O desenvolvimento foi dividido nas seguintes etapas:

3.1. Arquitetura Geral

O processador segue a arquitetura de caminho de dados monociclo, onde cada instrução é executada em um único ciclo de clock, passando pelas cinco etapas clássicas: IF, ID, EX, MEM e WB.

3.2. Módulos Implementados

Foram implementados em Verilog os seguintes módulos: ULA, Banco de Registradores (com tratamento para x0), Memória de Instruções (com programa pré-carregado), Memória de Dados, Unidade de Controle Principal (decodifica o opcode) e Unidade de Controle da ULA (decodifica functs). Um módulo de topo (`processador.v`) integra todos os componentes.

4. Análise de Resultados

A validação foi realizada com um programa de teste que utiliza todas as instruções implementadas. A análise a seguir combina a visualização das formas de onda no GTKWave com o log de saída do terminal gerado pelo testbench.

4.1. Execução da Instrução `lh` (Load Halfword / Tipo-I)

A instrução `lh x10, 0(x0)` é a primeira a ser executada. A forma de onda na Figura 1 ilustra a ativação dos sinais de controle `MemRead=1` e `RegWrite=1`. A ULA calcula o endereço (0), a memória lê o valor 10 (`0xA`), e o MUX `MemToReg` seleciona este valor para ser escrito em `x10`. A execução bem-sucedida é confirmada pelo log do terminal (Listagem 4.1).

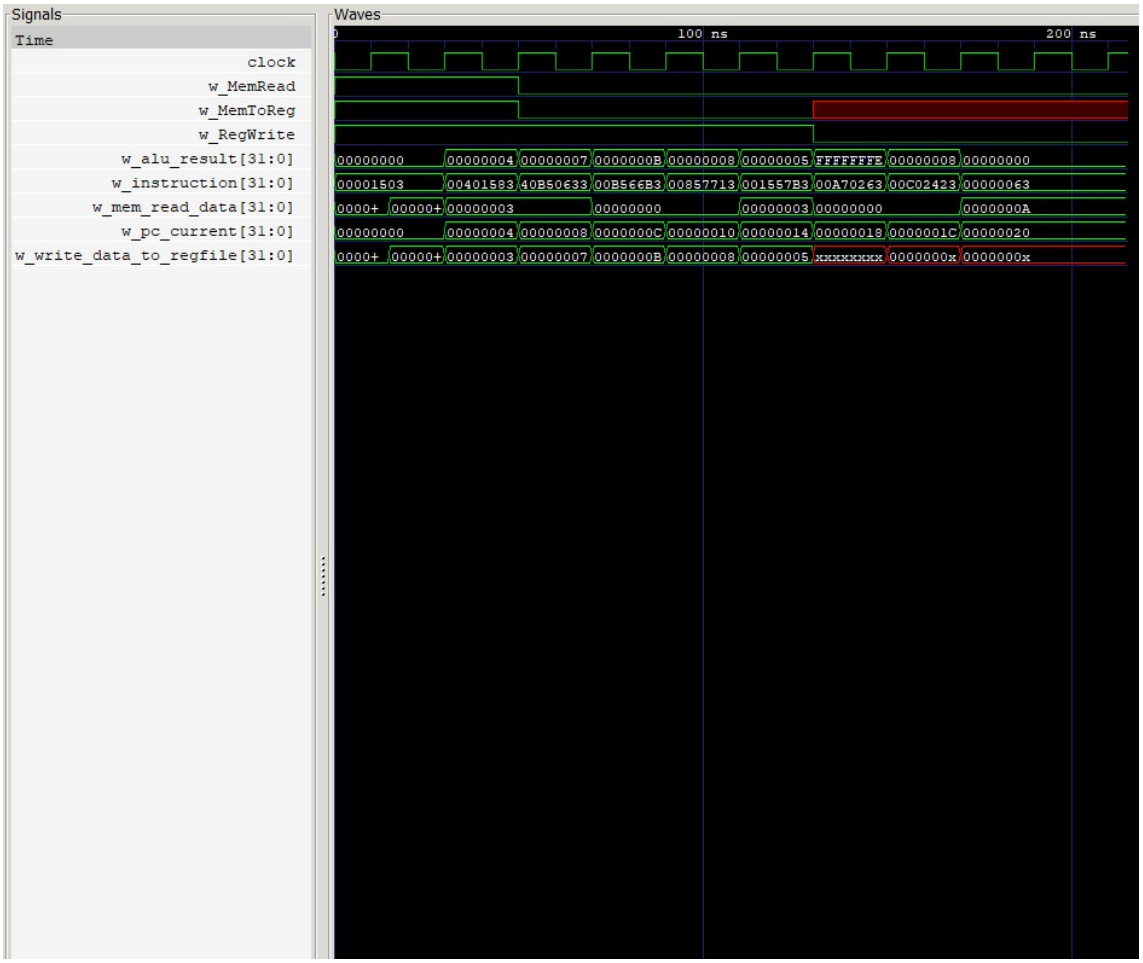

Figura 1. Execução da instrução `lh`: Sinais `MemRead` e `RegWrite` ativados para ler da memória e escrever no registrador.

```
>> PC=0x00, Executando lh x10, 0(x0)
   Lido da Mem[0] o valor          10. Escrito em x10.
   Resultado: x10 =          10
```

4.2. Execução da Instrução `sub` (Tipo-R)

Para a instrução `sub x12, x10, x11`, a Figura 2 mostra o sinal `ALUSrc=0`, indicando que os operandos vêm dos registradores (10 e 3). A ULA executa a subtração, e seu resultado (7) é selecionado pelo MUX `MemToReg` (`=0`) para ser escrito em `x12`, como validado pela Listagem 4.2.

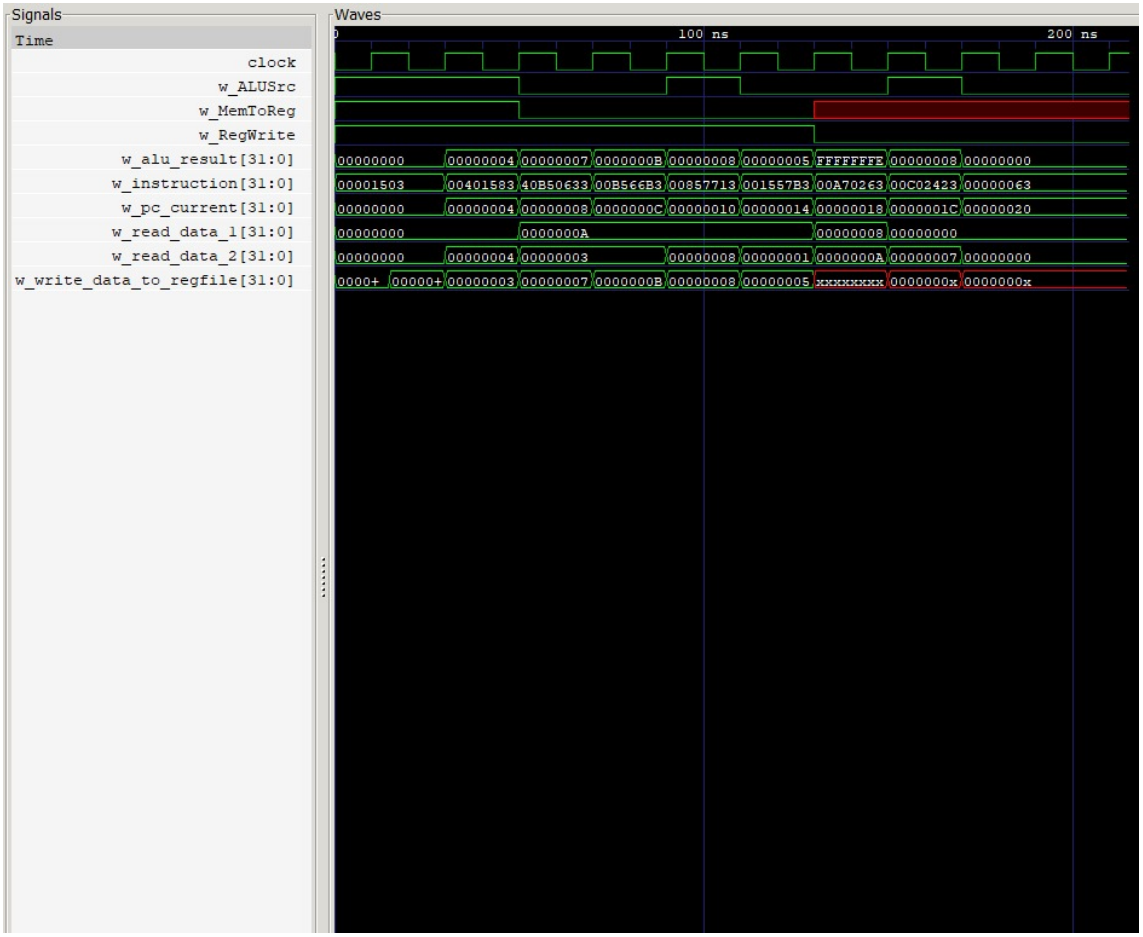

Figura 2. Execução da instrução `sub`: O sinal `ALUSrc` em 0 seleciona o operando do registrador `rs2` para a ULA.

```
>> PC=0x08, Executando sub x12, x10, x11
   Calculado          10 -           3. Escrito em x12.
   Resultado: x12 =           7
```

4.3. Execução da Instrução `sh` (Store Halfword / Tipo-S)

A instrução `sh x12, 8(x0)` armazena o valor do registrador `x12` (7) na memória. A Figura 3 mostra os sinais de controle cruciais para esta operação: `MemWrite=1` (habilita a escrita na memória) e `RegWrite=0` (desabilita a escrita no banco de registradores).

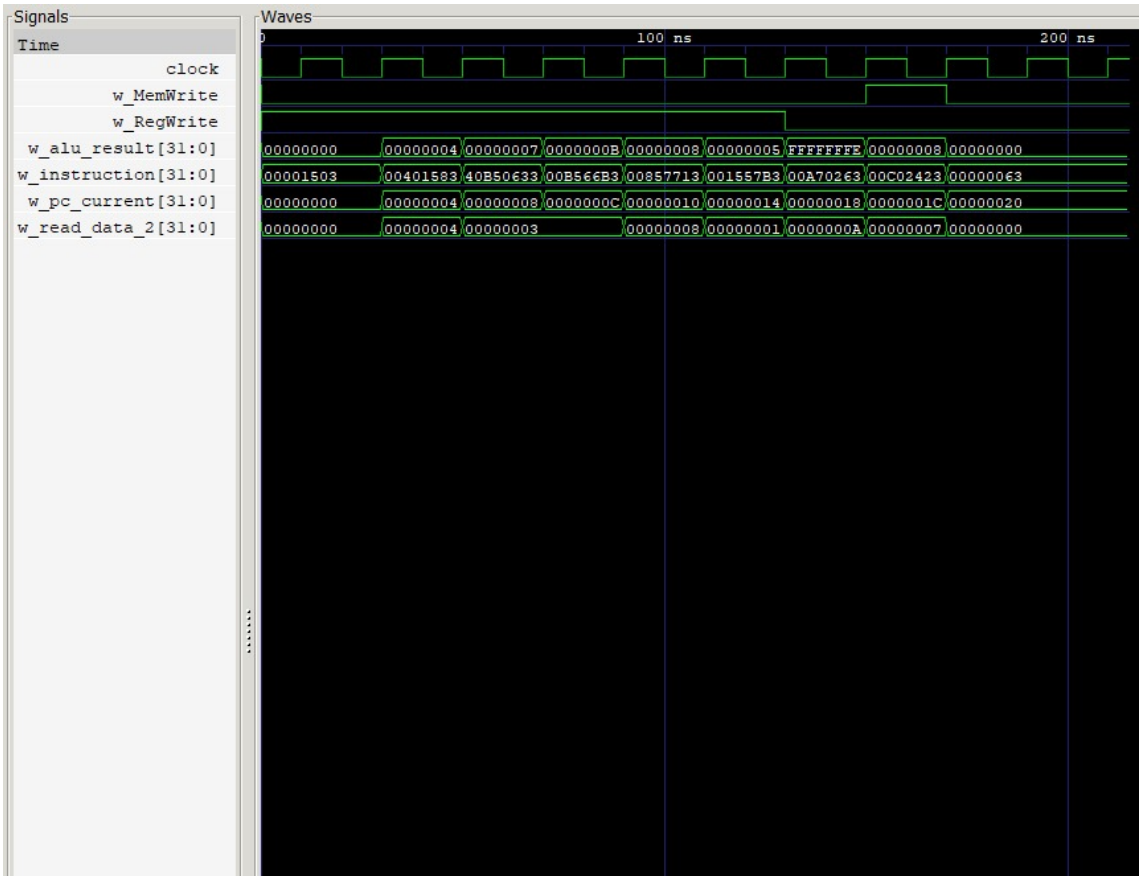

Figura 3. Execução da instrução `sh`: O sinal `MemWrite` é ativado enquanto o `RegWrite` permanece em 0.

```
>> PC=0x1C, Executando sh x12, 8(x0)
   Escrito o valor de x12(         7) no endereco
   de memoria 8. Verificacao no estado final.
```

4.4. Execução da Instrução `beq` (Branch Equal / Tipo-B)

A instrução `beq x14, x10, 4` compara os registradores `x14` (8) e `x10` (10). Como os valores são diferentes, a ULA não ativa a `zero_flag`. A Figura 4 mostra a `zero_flag=0`, o que faz com que o desvio não seja tomado e o PC receba `PC+4`, como confirmado na Listagem 4.4.

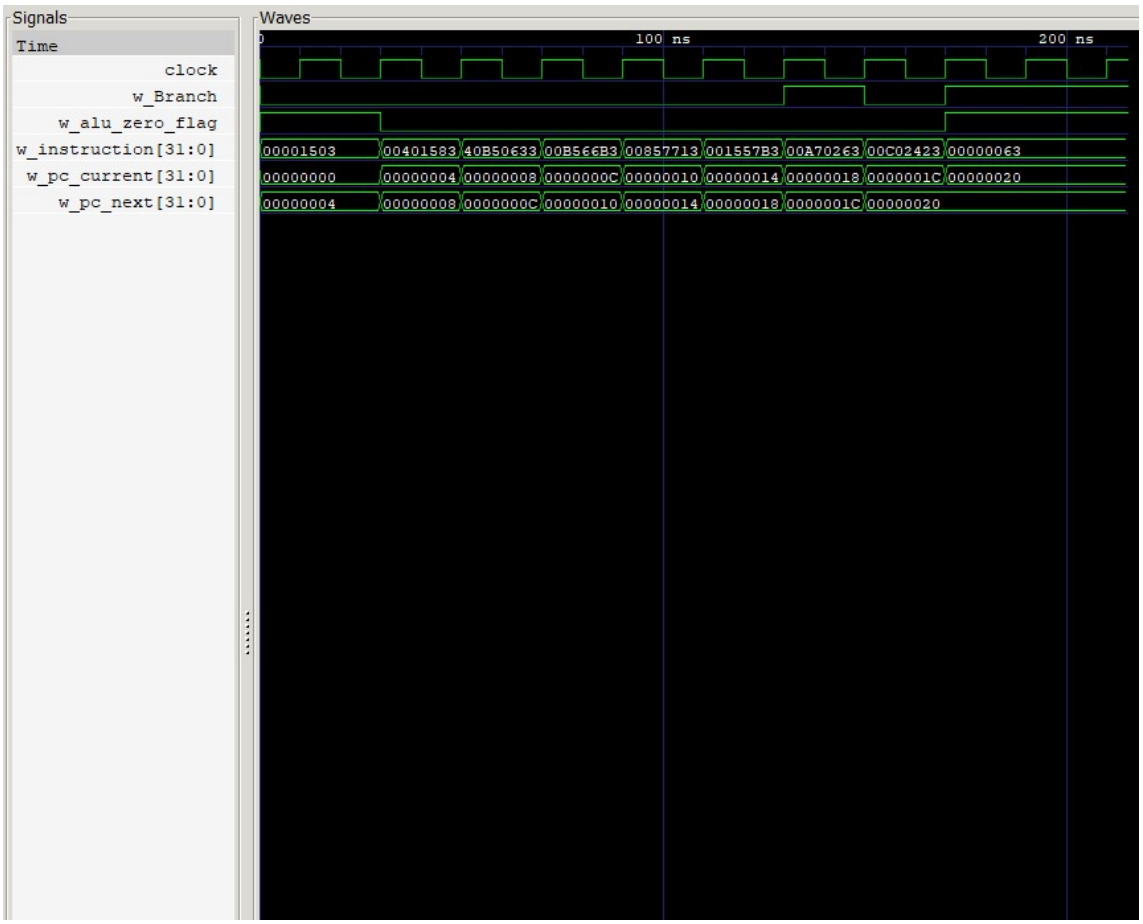

Figura 4. Execução do `beq` (desvio não tomado): A `zero_flag` em 0 impede o desvio.

```
>> PC=0x18, Executando beq x14, x10, 4
   Comparando x14(         8) e x10(        10). Como
   sao diferentes, o desvio NAO eh tomado.
```

4.5. Estado Final do Processador

Após a execução de todo o programa de teste, o estado final dos registradores e da memória de dados foi inspecionado (Listagem 1), confirmando que todos os valores esperados foram corretamente calculados e escritos, validando o projeto.

```
================ ESTADO FINAL ================

--- Registradores ---
x00: 0x00000000    x01: 0x00000001
x02: 0x00000002    x03: 0x00000003
x04: 0x00000004    x05: 0x00000005
x06: 0x00000006    x07: 0x00000007
x08: 0x00000008    x09: 0x00000009
x10: 0x0000000a    x11: 0x00000003
x12: 0x00000007    x13: 0x0000000b
x14: 0x00000008    x15: 0x00000005
x16: 0x00000010    x17: 0x00000011
x18: 0x00000012    x19: 0x00000013
x20: 0x00000014    x21: 0x00000015
x22: 0x00000016    x23: 0x00000017
x24: 0x00000018    x25: 0x00000019
x26: 0x0000001a    x27: 0x0000001b
x28: 0x0000001c    x29: 0x0000001d
x30: 0x0000001e    x31: 0x0000001f

--- Memoria de Dados (enderecos 0-36) ---
Mem[        0]: 0x0000000a
Mem[        4]: 0x00000003
Mem[        8]: 0x00000007
...

--- Flags ---
Zero Flag: 1
==========================================
```

Listagem 1. Estado final dos registradores e da memória de dados após a simulação.

5. Conclusão

A implementação do processador RISC-V monociclo possibilitou a consolidação de conceitos fundamentais de arquitetura de computadores, abrangendo desde o projeto e funcionamento da ULA até a integração harmoniosa de todos os módulos do sistema. O processo de desenvolvimento permitiu compreender em profundidade o fluxo de dados, a função de cada componente e a importância da coordenação entre as unidades de controle e execução.

A metodologia adotada, que combinou a análise detalhada das formas de onda no GTKWave com a interpretação dos registros gerados pelo testbench, mostrou-se eficiente para validar o correto funcionamento do processador em todas as instruções implementadas. Essa abordagem prática, aliada à simulação, contribuiu para identificar e corrigir possíveis inconsistências de forma rápida e precisa.

Como perspectivas futuras, destaca-se a possibilidade de evolução do projeto para uma arquitetura pipeline, visando ganhos de desempenho, bem como a ampliação do conjunto de instruções suportado, tornando o processador mais versátil e próximo de aplicações reais. Tais aprimoramentos não apenas aumentariam a complexidade e o desafio técnico, mas também reforçariam o entendimento de conceitos avançados em projeto e otimização de processadores.

Referências

[1] Patterson, D. A., Hennessy, J. L., *Computer Organization and Design RISC-V Edition*, Morgan Kaufmann, 2017.

[2] The RISC-V Instruction Set Manual, Volume I: User-Level ISA, Document Version 20191213.

[3] IEEE Standard for Verilog Hardware Description Language, IEEE Std 1364-2005.