
Universidade Federal de Campina Grande
Centro de Engenharia Elétrica e Informática
Departamento de Engenharia Elétrica
Projeto de Engenharia Elétrica

**Trabalho de Conclusão de Curso**

# Adaptação da arquitetura CVA6 RISC-V à extensão Bitmanip

Abdias Aires de Queiroz Neto

Campina Grande - PB
Dezembro de 2020

Abdias Aires de Queiroz Neto

# Trabalho de Conclusão de Curso

# Adaptação da arquitetura CVA6 RISC-V à extensão Bitmanip

*Trabalho de Conclusão de Curso submetido à Coordenação de Curso de Graduação de Engenharia Elétrica da Universidade Federal de Campina Grande como parte dos requisitos necessários para a obtenção do grau de Bacharel em Engenharia Elétrica.*

Área de Concentração: Eletrônica Digital.

Orientador: Prof. Gutemberg Gonçalves dos Santos Júnior, Dr. Sc

Campina Grande - PB

Dezembro de 2020

Universidade Federal de Campina Grande - UFCG
Centro de Engenharia Elétrica e Informática - CEEI
Departamento de Engenharia Elétrica - DEE
Projeto de Engenharia Elétrica

Abdias Aires de Queiroz Neto

# Trabalho de Conclusão de Curso

# Adaptação da arquitetura CVA6 RISC-V à extensão Bitmanip

Trabalho aprovado em: Campina Grande - PB, / /

---

**Abdias Aires de Queiroz Neto**
Aluno

---

**Gutemberg Gonçalves dos Santos Júnior**
Professor Orientador

Campina Grande - PB

Dezembro de 2020

# Resumo

Neste projeto, a arquitetura CVA6 RISC-V foi adaptada à extensão RISC-V Bitmanip. Uma nova unidade funcional foi construída a partir dos módulos de referência fornecidas pelo projeto RISC-V Bitmanip e inserida no estágio de execução do CVA6. Os estágios de decodificação e de leitura de operandos foram modificados para permitir o processamento do conjunto total de 103 instruções de manipulação de bits.

**Palavras-chave**: Processador, RISC-V, CVA6, Extensão Bitmanip, Sistemas Digitais

# Abstract

In this project, the CVA6 RISC-V architecture was adapted to the RISC-V Bitmanip extension. A new functional unit was designed based on the reference modules provided by the RISC-V Bitmanip project and inserted in the execute stage of CVA6. The instruction-decode and issue stages were modified to allow processing of the total set of 103 bitmanip instructions.

**Keywords**: Processor, RISC-V, CVA6, Bitmanip Extension, Digital Systems

# Lista de ilustrações

# Lista de tabelas

# Lista de abreviaturas e siglas

| | |
|---|---|
| RISC | *Reduced Instruction Set Computer* |
| ISA | *Instruction Set Architecture* |
| Bitmanip | *Bit Manipulation* |

# Sumário

# 1 Introdução

Operações de manipulação de bits são empregadas rotineiramente em aplicações de criptografia, processamento de imagem e compressão de dados. A verstalidade dessa classe de operações se deve a que elas manipulam dígitos binários de um operando de forma individual, em vez de os considerarem em conjunto como é feito em uma adição ou uma multiplicação, por exemplo. Apesar de seu uso corriqueiro em processadores, poucas operações *bitmanip* têm representação direta em arquiteturas de conjunto de instruções (ISA - *Instruction Set Architectures*), embora em alguns casos, como o das arquiteturas x86 da Intel e da AMD, sejam fornecidas em extensões.

Quando a manipulação de bits é necessária e uma extensão não está disponível, a operação é mapeada em uma sequência de duas ou mais instruções básicas com comportamento equivalente. Comparado a uma representação direta no ISA, o mapeamento indireto apresenta o inconveniente de prolongar o tempo de execução de tarefas e de elevar o consumo de energia do processador.

Uma extensão Bitmanip para o conjunto RISC-V tem sido desenvolvida por um dos grupos de trabalho que se dedicam ao aprimoramento dessa ISA. Na fase atual do projeto, um conjunto de 103 de instruções encontra-se selecionado e teve seu comportamento definido. Além disso, foram propostas codificações binárias provisórias e implementações de referência em *hardware*. Apesar de inacabado, o projeto já permite realizações experimentais em processadores.

## 1.1 Objetivo

O objetivo deste trabalho é implementar a extensão Bitmanip na arquitetura RISC-V CVA6, antiga Ariane. Esse processador, que foi desenvolvido para aplicações de uso geral, é capaz de executar o sistema operacional Linux nos três níveis usuais de privilégio, e já foi testado em aplicações industriais e acadêmicas. Além disso, tem características favoráveis a uma adaptação de arquitetura: é codificada diretamente em linguagem de descrição de *hardware* (*SystemVerilog*), em contraste com sistemas como o Rocket, de Berkeley; e está disponível publicamente na plataforma GitHub. Assim, serão efetuadas adaptações na estrutura do processador para torná-lo capaz de processar as 103 novas instruções *bitmanip*.

Este projeto é uma continuação do trabalho introdutório realizado pelo autor em um estágio no laboratório Embedded, da Universidade Federal de Campina Grande, entre janeiro e fevereiro de 2020. Nesse trabalho, um subgrupo de 6 instruções *bitmanip*, do tipo

contagem de bits, foi implementado no processador CVA6, então chamado Ariane.

# 2 A arquitetura RISC-V CVA6 e a extensão Bitmanip

A arquitetura CVA6 RISC-V, chamada Ariane até 2020, foi desenvolvida por uma colaboração de pesquisa entre a ETH Zurique e o grupo PULP, ligado à Universidade de Bolonha. Trata-se de um processador RISC-V que implementa as extensões I, M, A e C da ISA para 64 bits, e é capaz de executar um sistema operacional do tipo Unix nos níveis de privilégio M, S e U. Seu *pipeline* tem seis estágios e executa instruções de forma ordenada (*in-order*) e *single-issue* (ZARUBA, 2018b).

Essa arquitetura tem características que viabilizam sua extensão a novas funções. É disponibilizada em código aberto na plataforma *GitHub*, tendo sido atualizada regularmente desde sua publicação original em 2018. Em contraste com arquiteturas como a Rocket, da Berkeley, o *hardware* do CVA6 é descrito diretamente em *SystemVerilog*, o que permite alterações localizadas na estrutura. Além disso, esse processador foi projetado para ir além de aplicações experimentais, tendo sido empregado em um número de projetos acadêmicos e de indústria (ZARUBA; BENINI, 2019). É usado, por exemplo, na *European Processor Initiative* em sistemas de cálculo de escala extrema (EPI, 2020).

O *pipeline* do CVA6, apresentado na figura 1, tem 6 estágios com funções semelhantes às encontradas em outras arquiteturas RISC (ZARUBA, 2018a). Neste projeto, todas as alterações foram feitas nos estágios de decodificação, leitura de operandos e execução. Em particular, como o estágio de execução no CVA6 apresenta forte modularidade, toda a modificação efetuada nesse componente consistiu em se introduzir uma nova unidade funcional para operações *bitmanip*, a ser descrita na seção seguinte.

## 2.1 A extensão RISC-V Bitmanip

A extensão Bitmanip é projetada por um dos grupos de trabalho das fundações RISC-V. O objetivo desse projeto é selecionar as operações de manipulação de bits a ser implementadas como instruções do conjunto. Além disso, o projeto especifica precisamente comportamento e a codificação binária das instruções (o que inclui seus *opcodes*). Também são fornecidas unidades de cálculo de referência em Verilog e uma infra-estrutura de testes, que inclui programas de validação em *assembly* e ferramentas de compilação e simulação para RISC-V (WOLFF, 2019b).

Os critérios de seleção de instruções *bitmanip* são os seguintes (WOLFF, 2019a):

- consistência arquitetural: as mudanças no ISA devem desviar o mínimo possível dos

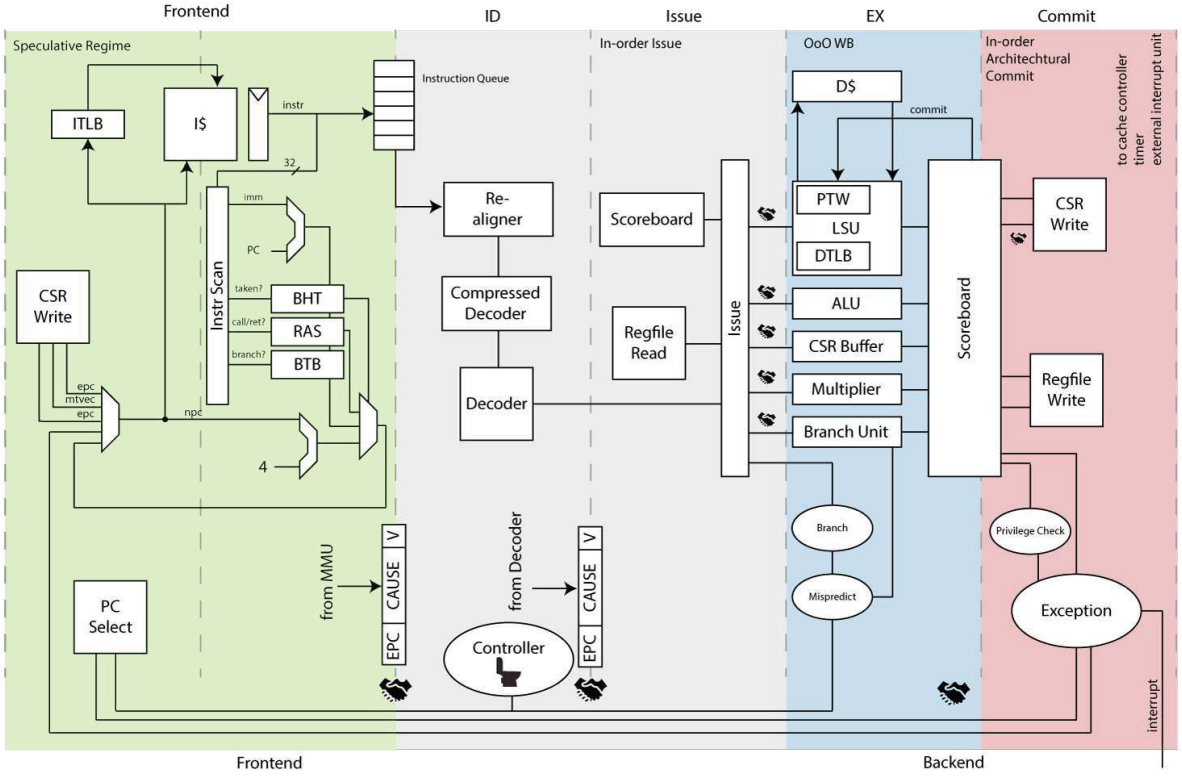
padrões existentes e não devem implementar novamente operações já presentes no conjunto;

- eficiência: as mudanças devem resultar em ganhos significativos de ciclos de *clock* ou de número de instruções em programas; em geral, cada instrução *bitmanip* deverá ser capaz de subsituir ao menos 3 instruções já existentes;
- aplicatividade: as mudanças devem mostrar aumento de desempenho de programas em aplicações reais, validadas por testes de *benchmark*;
- simplicidade de *hardware*: propostas que impliquem grande aumento de complexidade de *hardware* devem ser desfavorecidas.

Os resultados do projeto são disponibilizados na plataforma *GitHub*. Embora ainda em curso, o trabalho se encontra em estágio avançado: o grupo de instruções a ser incluídas no conjunto já foi definido e teve comportamento e codificação binária já especificados, embora os *opcodes* sejam considerados provisórios. Em relação à data de publicação deste relatório, a atualização mais recente do projeto havia sido feita em outubro de 2020.

O conjunto proposto tem um total de 103 instruções. Como as instruções são organizadas em subconjuntos (figura 2), não será preciso que cada arquitetura de processador

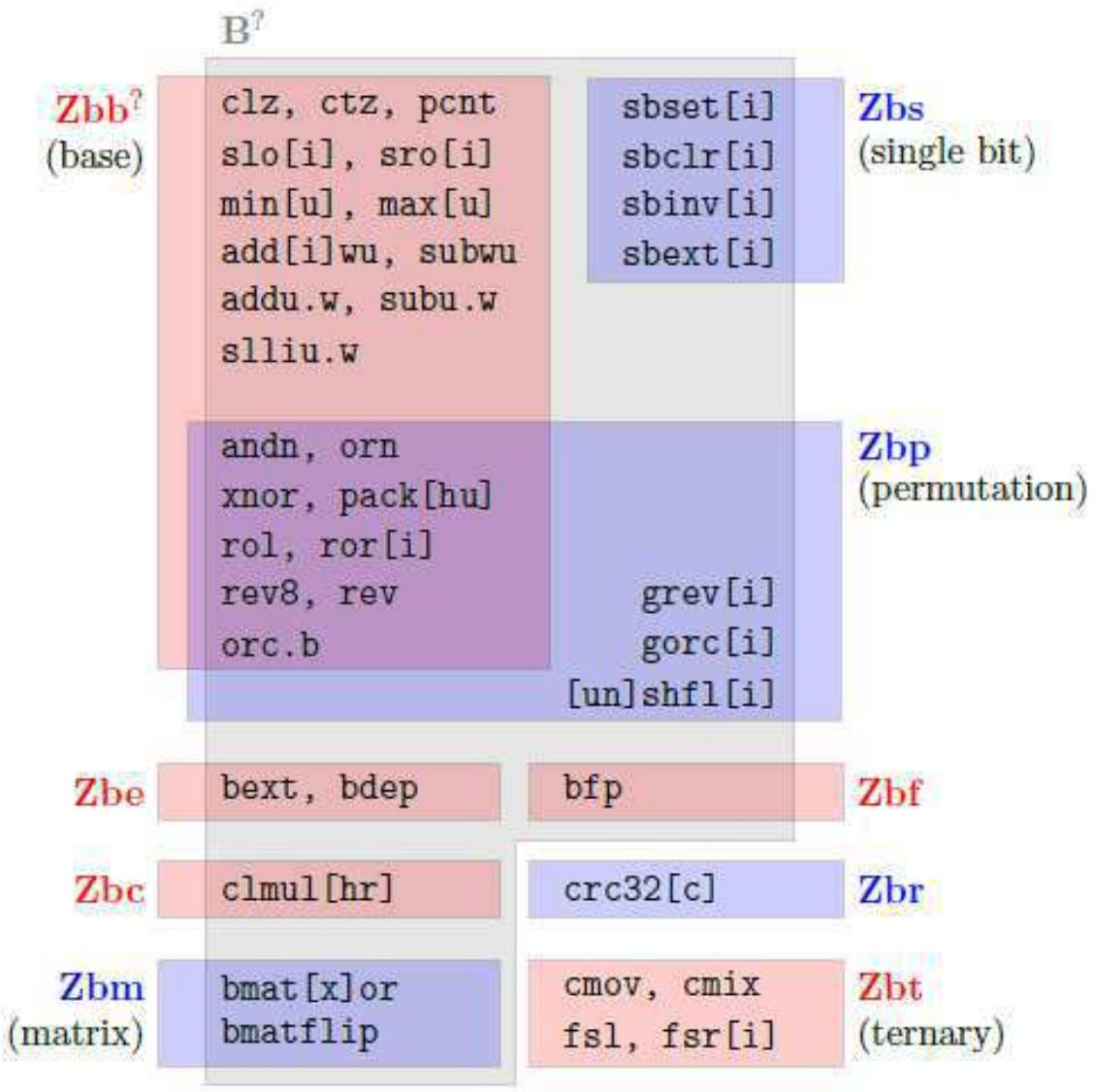
Figura 1 – *Pipeline* do processador RISC-V CVA6




Fonte: `https://github.com/abdiasaires/ariane-bitmanip`.

implemente a extensão completa - o que resultaria em aumento significativo de *hardware* -, mas apenas um ou mais subconjuntos disponíveis, a depender da aplicação. Neste projeto, porém, o CVA6 foi estendido ao grupo total de 103 instruções.

Figura 2 – Proposta de extensão RISC-V bitmanip.




Fonte: `https://github.com/riscv/riscv-bitmanip/bitmanip-092.pdf`.

# 3 Adaptação da arquitetura

## 3.1 A unidade *bitmanip*

Todas as operações *bitmanip* são executadas em uma única unidade funcional, chamada unidade *bitmanip*, inserida no estágio de execução do CVA6. Essa unidade é composta dos módulos de referência fornecidos pelo projeto RISC-V Bitmanip e implementados em Verilog. Cada módulo é capaz de executar uma parte do conjunto total de instruções:

- módulo *bextdep*: instruções de permutação de bits, como SHFL (*shuffle*) e GREV (*generalized revolution*);
- módulo *bitcnt*: instruções de contagem de bits, como PCNT (*population count* - contagem de bits 1);
- módulo *bmatxor*: instruções de manipulação de matrizes, como BMATXOR (multiplicação de matrizes);
- módulo *clmul*: instruções de multiplicação;
- módulo *crc*: instruções de redução polinomial, como CRC32.B (*cycle redundancy check*);
- módulo *shifter*: instruções de deslocamento, como FSR (*funnel shift*);
- módulo *simple*: instruções semelhantes às do conjunto básico, como ANDN.

Além das novas instruções especificadas pelo projeto, os módulos de referência implementam também algumas das instruções já presentes no conjunto básico, como os deslocamentos lógicos (SRL e SLL). A razão disso é permitir maior uniformidade de processamento, com instruções semelhantes (p. ex., todos os deslocamentos) executadas na mesma unidade. Por esse motivo, no decodificador do CVA6, certas instruções normalmente executadas pela ULA passaram a ser atribuídas à nova unidade *bitmanip*.

Para compor essa unidade, os módulos de referência foram introduzidos em um *wrapper*. A função do *wrapper* é, em primeiro lugar, adaptar os sinais de interface dos módulos de cálculo àqueles presentes no ambiente CVA6. A tabela 1 apresenta os sinais de interface de um módulo de cálculo típico, e a 2 exibe a interface da unidade *bitmanip*. Os argumentos da operação estão contidos no sinal *fu data*, descrito na tabela 3.

Tabela 1 – Interface do módulo *bitcnt*

| **Sinal** | **Descrição** |
|---|---|
| Clock, reset<br>(in) | Sinais de controle |
| Data in valid, data in ready<br>(in) | Handshake de entrada |
| RS1<br>(in) | Operando |
| Inst3, inst20, inst21,<br>inst22<br>(in) | Código da operação (4 bits) |
| Data out<br>(out) | Resultado |
| Data out valid, ready<br>(out) | Handshake de saída |

Tabela 2 – Interface da unidade de *bitmanip*

| **Sinal** | **Descrição** |
|---|---|
| Clock, reset, flush<br>(in) | Sinais de controle |
| FU data<br>(in) | Operação a se executar |
| Bitmanip valid<br>(in) | Operação é válida |
| Bitmanip ready<br>(out) | Unidade bitmanip pode receber nova operação |
| Bitmanip result<br>(out) | Resultado da operação para write-back |
| Bitmanip transaction ID<br>(out) | Entrada no scoreboard do resultado |
| Bitmanip valid<br>(out) | Resultado é válido |

O procolo de *handshake*, que ocorre na ordem *valid-ready* na entrada dos módulos de cálculo, é invertido pelo *wrapper* na interface com o CVA6. Além disso, o *wrapper* mantém controle do *transcation ID* da operação, que é fornecido juntamente com o resultado do cálculo e é necessário para escrita (*write-back*) na posição correta da fila de instruções.

O *wrapper* decodifica o código da operação recebida a fim de encaminhá-la ao módulo de cálculo correto. Essa etapa é necessária porque o estágio de leitura de operandos do CVA6 identifica cada operação com um código particular (trecho *operator* do sinal *fu data*, na tabela 3), que não guarda relação direta com o código binário da instrução - este não é fornecido às unidades funcionais. Assim, foi necessário introduzir na unidade um decodificador de tamanho significativo, dado o número de operações possíveis. Um mecanismo análogo é adotado em outras unidades funcionais do CVA6, como a FPU.

Tabela 3 – Interface de entrada de operações da unidade *bitmanip*

| **Sinal** | | **Descrição** |
|---|---|---|
| FU data (estrutura) (out) | FU | Código da unidade funcional |
| | Operator | Código da operação |
| | Operands A, B | Operandos |
| | Imm | Operando imediato |
| | Transaction ID | Entrada no scoreboard |
| Input valid (out) | | Operação é válida |
| FU ready (in) | | UF pode receber operação |

### 3.1.1 Processamento de operações na unidade *bitmanip*

Se for tomado como exemplo o módulo *bitcnt*, cuja interface já foi apresentada na tabela 1, o processo de execução na unidade *bitmanip* poderá ser descrito da seguinte forma. Uma operação do tipo contagem de bits, como PCNT, chega à interface da unidade a partir do estágio de leitura de operandos. Nessa etapa, a operação encontra-se identificada por seu código de operação, *BM PCNT*. O decodificador da unidade identifica o módulo de cálculo a ser empregado na execução (*bitcnt*) e fornece os sinais seletores da operação PCNT adequados à interface desse módulo - nesse caso, os bits *inst3*, *inst20*, *inst21* e *inst22* da tabela 1. Após a execução, o resultado é apresentado na saída da unidade juntamente com seu *transaction ID*.

Devido à introdução de um decodificador e à complexidade de *hardware* de alguns dos módulos de cálculo, a unidade *bitmanip* foi implementada com um *pipeline* de três estágios: decodificação, processamento e saída. O processamento, por sua vez, ocorre em mais de um ciclo em alguns dos módulos. Essa estrutura é semelhante à do módulo *full*, que contém os demais módulos e foi fornecida pelo projeto - mas com interface e decodificação incompatíveis com o CVA6.

Uma limitação da unidade implementada é a ausência de paralelização entre os módulos de cálculo. Dado o grande número de operações que podem ser realizadas na unidade *bitmanip* - o que inclui manipulações básicas como os deslocamentos -, essa limitação pode resultar em ganhos apreciáveis de ciclos de *clock* na execução de programas. Por outro lado, esse problema pode ser solucionado por uma sofisticação do sistema de *handshake* da própria unidade *bitmanip*, de modo similar ao adotado na FPU do CVA6 - isto é, a solução do problema não implica mudanças da arquitetura circundante. Uma das premissas deste projeto é que a unidade funcional é provisória e deverá ser substituída por um componente especificamente projetado para o CVA6 - o que já estava implícito na escolha dos módulos de referência como unidades de cálculo. Assim, a eventual substituição desse componente não tornará necessárias outras adaptações da arquitetura CVA6.

Tabela 4 – Sinais de saída do decodificador

| **Sinal** | **Descrição** |
|---|---|
| PC | Endereço virtual da instrução |
| FU | Unidade funcional |
| OP | Operação |
| RS1, RS2 | Registros de operandos |
| RD | Registro de destino |
| Result | Resultado |
| Use I Immediate | Use o operando imediato como segundo operando |
| Use Z Immediate | Use operando imediato Z como primeiro operando |
| Use PC | Use o PC como operando |
| Branch predict | Estrutura de dados para predição de branches |
| Is compressed | A instrução é comprimida |

## 3.2 Adaptação do decodificador

O decodificador do CVA6 produz os sinais de saída apresentados na tabela 4. Eles permitem identificar, por exemplo, a operação a ser executada, seus operandos e (quando apropriado) o registro de destino.

Foi introduzida no decodificador a lógica para identificação das 103 instruções *bitmanip*, conforme codificadas pelo projeto, além das instruções do conjunto básico também executadas na nova unidade. Tentou-se utilizar ao máximo as estruturas de seleção já presentes no decodificador. A modificação mais importanto foi introduzida para tratamento de instruções ternárias (de três operandos), dado que elas não tinham correspondência direta com instruções conhecidas do decodificador (ver exemplos na figura 3). Especificamente, foram introduzidos dois novos modos de seleção de imediato:

- para instruções ternárias com três registros de operando (como FSR), o decodificador no modo *RS3* coloca o endereço dos dois primeiros registros nos campos *rs1* e *rs2* (tabela 4) e o endereço do terceiro registro no campo *result*;
- para instruções ternárias com dois registros e um imediato (como FSRI), o modo *BMRS3* atribui o imediado ao campo *result*.

Figura 3 – Instruções *bitmanip* ternárias.

```
|   rs3   | 10|   rs2   |   rs1   | 101 |    rd   |   0110011   |  FSR
|   rs3   |1|    imm    |   rs1   | 101 |    rd   |   0010011   |  FSRI
|-------------------------------------------------------------|
```

Fonte: `https://github.com/riscv/riscv-bitmanip/bitmanip-092.pdf`.

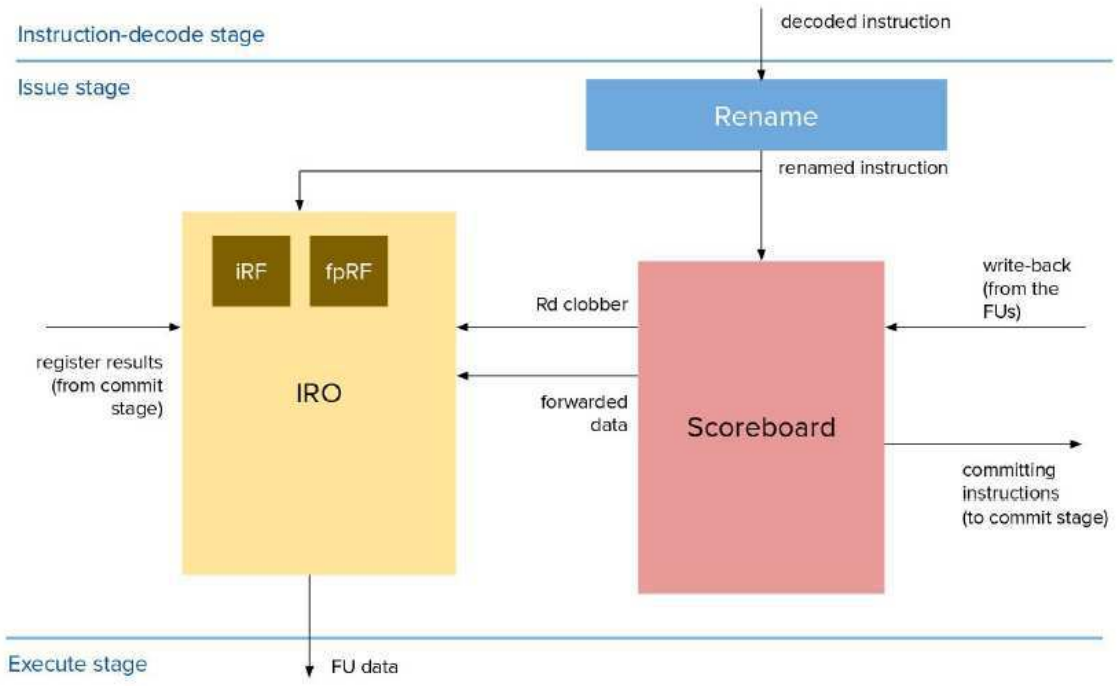
## 3.3 Adaptação do estágio de leitura de operandos

O estágio de leitura de operandos da arquitetura CVA6 tem a estrutura apresentada na figura 4. O banco de registros pertence ao módulo *issue and read operands* (IRO), que é responsável pela seleção de operandos. O módulo *scoreboard* é uma fila de instruções que também armazena os resultados de *write-back*, antes de sua transferência ao banco de registros.

A adaptação desse estágio consitiu, em primeiro lugar, na inserção de uma interface para comunicação com a unidade *bitmanip* (já apresentada na tabela 3), que seguiu a estrutura usada em outras unidades funcionais.

Além disso, foi adicionada a lógica para manipulação de operações ternárias. O banco de registros foi modificado para ter três portas, o que foi obtido por ajuste de um parâmetro e por extensão dos *arrays* de endereços de entrada e de dados de leitura. Se uma operação bitmanip do tipo ternária é recebida, o módulo faz leitura dos três registros e posiciona os operandos nos trechos $A$, $B$ e $Imm$ da interface com a unidade. Se essa operação tem um imediato com operando (casos de FSRI e FSRIW, unicamente), este é posicionado no trecho $B$, de modo a seguir a codificação especificada no projeto.

Figura 4 – Estágio de leitura de operandos.



# 4 Validação e resultados

Para validar de forma eficiente a adaptação da arquitetura, a estratégia de testes adotada foi a seguinte: admitiu-se que os módulos de cálculo funcionavam corretamente e concentrou-se o esforço de verificação sobre a decodificação de instruções, a leitura de operandos e a escrita dos resultados. Isto é, os sinais observados foram os da saída do decodificador e os da saída do estágio de leitura de operandos. Também foi assegurado que cada programa de teste era executado até o fim sem causar bloqueios no *pipeline* - o que teria indicado, por exemplo, que a unidade *bitmanip* havia produzido sinais de saída com *transaction ID* incorreto.

Dado o grande número de instruções (superior a 103) a se verificar, essa metodologia teve a vantagem de que cada instrução pôde ser testada com um único exemplo no programa de teste, sem que houvesse a preocupação de fornecer variações de seus operandos - o que teria sido necessário se o correto funcionamento dos módulos de cálculo não estivesse admitido por hipótese. Por outro lado, a observação das saídas dos estágios de decodificação e de leitura de operandos foi suficiente verificar o funcionamento dos únicos estágios do CVA6 que sofreram adaptações de arquitetura, e a observação de que os programas chegavam ao fim corretamente permitiu garantir que o *wrapper* da unidade *bitmanip* encaminhava as operações sem causar bloqueios.

Por outro lado, a metodologia adotada tem limitações. Ela não permite testar o gerenciamento de dependências de dados ou o correto processamento de instruções do conjunto básico - isto é, não inclui testes de não-regressão. São feitas sugestões para testes mais incisivos ao fim desta seção.

## 4.1 Aplicação da metodologia

Foi empregado o compilador RISC-V fornecido pelo projeto RISC-V Bitmanip. Esse compilador contém os *opcodes* de todas as instruções propostas no projeto.

A verificação detalhada do processador requeriu observação de seus sinais internos. Foi empregado o *software* Modelsim para simulação da arquitetura e visualização das formas de onda. Em particular, deu-se atenção às saídas do decodificador e do estágio de leitura de operandos, como exemplificado na figura 5. O programa de testes, escrito em *assembly*, é compilado e convertido em um arquivo de memória, que em seguida é carregado na memória RAM da plataforma de teste.

A aplicação dessa metodologia permitiu validar, dentro de seus limites, todas as instruções introduzidas no decodificador do CVA6. A subseção seguinte contém sugestões

para uma validação mais ampla.

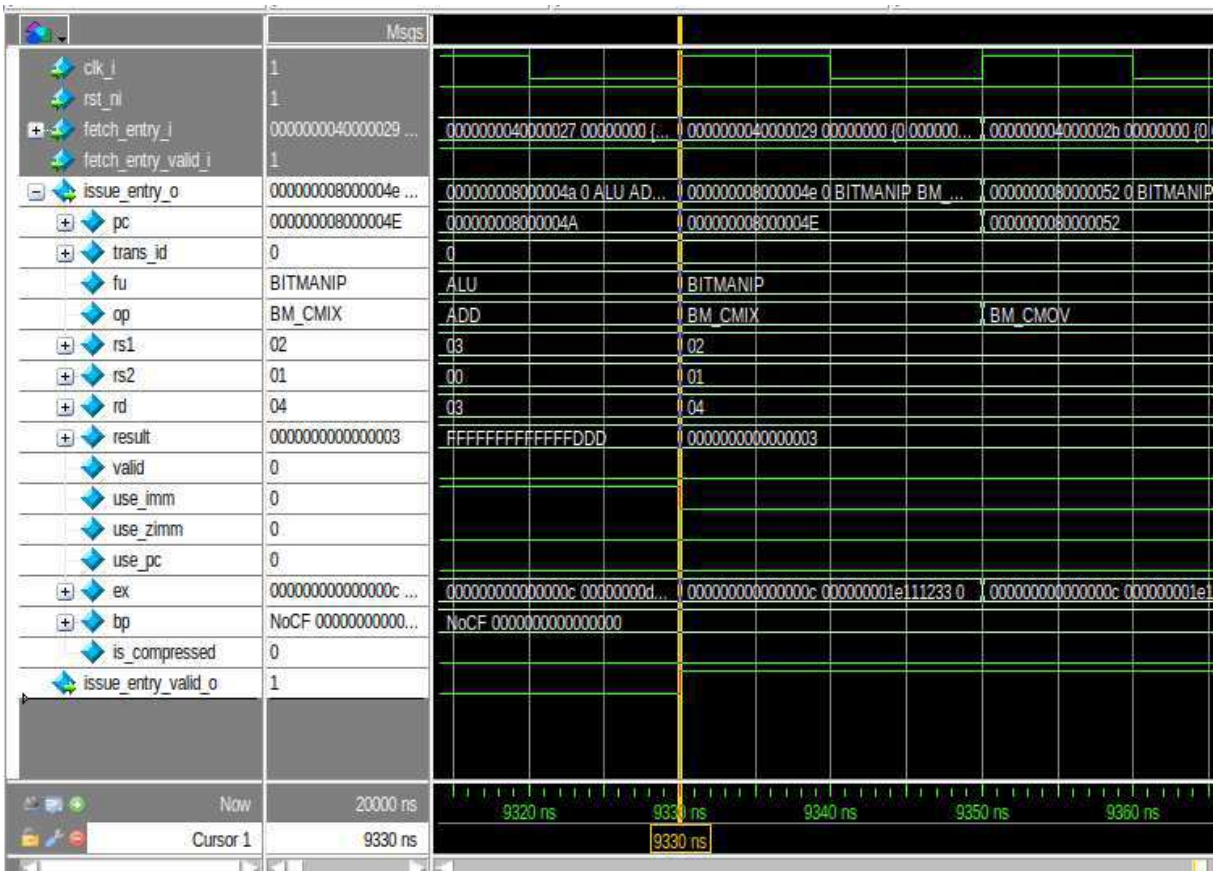
## 4.2 Sugestões para validação futura

Como a adaptação exigiu alterações no decodificador e na lógica de leitura de operandos, é sugerida a aplicação de testes de não-regressão - isto é, testes da arquitetura para a funcionalidade do sistema original. Um exemplo de testes desse tipo é o conjunto *torture*, fornecido com ferramentas de simulação RISC-V.

Também é sugerida a validação do gerenciamento de dependências de dados, especialmente para as instruções ternárias. Isso pode ser feito com a elaboração de programas com instruções *bitmanip* nas sequências *read-after-write* (RAW) e *write-after-read* (WAR).

Uma validação mais incisiva da unidade *bitmanip* pode ser feita por meio da infraestrutura de teste já disponibilizada pelo projeto Bitmanip. Esse sistema contém testes em *assembly* para cada instrução e compara as saídas da simulação com os resultados de programas de referência escritos em C, que implementam os comportamentos esperados. No entanto, esse sistema é construído para simulações com o módulo *spike* e deverá ser

Figura 5 – Formas de onda na saída do decodificador.



adaptado ao teste do CVA6 em uma ferramenta como o Modelsim.

# 5 Conclusão

Neste projeto, a arquitetura CVA6 RISC-V foi adaptada à extensão RISC-V Bitmanip. As modificações da arquitetura se concentraram nos estágios de decodificação e de leitura de operandos. O decodificador foi estendido para detectar as 103 novas instruções, conforme a condificação binária proposta pelo projeto RISC-V Bitmanip. Foram introduzidos dois novos modos de seleção de imediato para tratamento de instruções ternárias.

No estágio de execução, foi acrescentada uma interface para comunicação com uma nova unidade de cálculo. O banco de registros foi modificado para ter três portas, e foi inserida a lógica de tratamento de operações ternárias.

Uma nova unidade funcional foi introduzida no estágio de execução. Ela foi construída a partir dos módulos de referência fornecidos pelo projeto, inseridos em um *wrapper* para adequação de sua interface ao ambiente CVA6. O *wrapper* também é responsável pela decodificação das operações recebidas do estágio anterior e por seu encaminhamento aos módulos de cálculo.

A validação do sistema consistiu em verificar a decodificação de instruções, a leitura dos respectivos operandos e o funcionamento sem bloqueios da unidade *bitmanip*. Admitiu-se por hipótese que os módulos de cálculo fornecidos por referência funcionavam como esperado. São feitas sugestões para uma validação mais ampla no futuro, como por testes de não-regressão e de dependência de dados.

# Bibliografia

EPI. *European Processor Initiative*. [S.l.]: Acesso: `https://www.european-processor-initiative.eu/project/epi/`, 2020. Citado na página 11.

WOLFF, C. *Bitmanip RISC-V Extension*. [S.l.]: Acesso: `https://github.com/riscv/riscv-bitmanip/bitmanip-0.92.pdf`, 2019. Citado na página 11.

WOLFF, C. *RISC-V Bitmanip (Bit Manipulation) Extension*. [S.l.]: Acesso: `https://github.com/riscv/riscv-bitmanip`, 2019. Citado na página 11.

ZARUBA, F. *Ariane*. [S.l.]: Acesso: `https://pulp-platform.github.io/ariane/docs/home/`, 2018. Citado na página 11.

ZARUBA, F. *Ariane RISC-V CPU*. [S.l.]: Acesso: `https://github.com/pulp-platform/ariane`, 2018. Citado na página 11.

ZARUBA, F.; BENINI, L. The cost of application-class processing: Energy and performance analysis of a linux-ready 1.7-ghz 64-bit risc-v core in 22-nm fdsoi technology. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, v. 27, n. 11, p. 2629–2640, Nov 2019. ISSN 1557-9999. Citado na página 11.