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

Relatório de Estágio Supervisionado
Laboratório Xmen

Adaptação da arquitetura Ariane RISC-V a instruções de manipulação de bits

Abdias Aires de Queiroz Neto

Campina Grande - PB
Fevereiro de 2020

Abdias Aires de Queiroz Neto

Relatório de Estágio Supervisionado
Laboratório Xmen

Adaptação da arquitetura Ariane RISC-V a instruções de manipulação de bits

Relatório de Estágio Supervisionado 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.

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

Campina Grande - PB

Fevereiro 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

Relatório de Estágio Supervisionado
Laboratório Xmen

Adaptação da arquitetura Ariane RISC-V a instruções de manipulação de bits

Trabalho aprovado em: Campina Grande - PB, / /

Abdias Aires de Queiroz Neto
Aluno

Gutemerg Gonçalves dos Santos Júnior, Dr. Sc
Professor Orientador

Campina Grande - PB

Fevereiro de 2020

Resumo

O presente relatório descreve as atividades realizadas por Abdias Aires de Queiroz Neto, estudante de Engenharia Elétrica na Universidade Federal de Campina Grande, durante um estágio no Laboratório Embedded/Xmen, entre 9 de Janeiro e 11 de Fevereiro de 2020. O trabalho consistiu em adaptar a arquitetura do processador Ariane RISC-V a um subconjunto de instruções da extensão RISC-V Bitmanip, que estava em fase de desenvolvimento.

Palavras-chave: Ariane, RISC-V, Bitmanip, processador, *design* digital.

Abstract

This report describes the activities executed by the intern Abdias Aires de Queiroz Neto, student of Electrical Engineering at Universidade Federal de Campina Grande, during an internship performed at the Embedded/Xmen Laboratory between January 9th and February 11th of 2020. The objective of the internship was to adapt the Ariane RISC-V processor's architecture to make it capable of processing instructions of the RISC-V Bitmanip extension, which was in state of development.

Keywords: Ariane, RISC-V, Bitmanip, processor, digital design.

Lista de ilustrações

Lista de tabelas

Lista de abreviaturas e siglas

RISC	*Reduced Instruction Set Computer*
Bitmanip	*Bit manipulation*
UF (FU)	*Unidade funcional)* (Functional unit)
IRO	*Issue and read operands (module)*

Sumário

1 Introdução

Acelerar a execução de programas sempre foi uma das metas centrais da indústria eletrônica. Executar as mesmas tarefas em menos tempo permite processar mais dados em um mesmo intervalo. Quando a redução do tempo é obtida por redução do número de operações (e, portanto, por abaixamento da atividade elétrica do circuito), essa economia resulta também em menor consumo de energia.

Para processadores com conjuntos de instruções já estabelecidos, a redução do número de operações pode ser obtida por inclusão de novas instruções no conjunto, de modo que cada instrução acrescentada permita substituir duas ou mais instruções convencionais em certa operação. O resultado é mais vantajoso quando a operação é de uso frequente em aplicações do mundo real.

Operações ditas de "manipulação de bits", ou de bitmanip, enquadram-se nessa categoria. Trata-se de operações que agem individualmente sobre os dígitos binários de um operando, ao invés de tratá-los em conjunto como o faz uma adição convencional, por exemplo. Exemplos de operações bitmanip incluem os deslocamentos à esquerda ou à direita e operações lógicas como *and* e *or*.

Embora um número de operações bitmanip já estejam incluídas no conjunto RISC-V, argumenta-se que o desempenho de processadores seria afetado positivamente se mais instruções desse tipo estivessem disponíveis. O grupo de trabalho RISC-V Bitmanip, das Fundações RISC-V, tem tentado suprir essa demanda (WOLFF, 2019a). O objetivo do projeto é especificar um conjunto de instruções bitmanip como nova extensão do RISC-V. Embora ainda em curso, o projeto está em estado avançado de desenvolvimento, e já especificou o comportamento de 42 novas instruções.

Resta implementá-las em processadores RISC-V de uso corrente. Embora essas instruções estejam sujeitas a mudanças quanto à codificação, uma primeira implementação experimental seria vantajosa por dois motivos: (1) permitiria antecipar as adaptações a ser empreendidas nas arquiteturas quando da implementação definitiva; e (2) permitiria testar em um estágio adiantado o ganho em desempenho promovido pelas novas instruções.

1.1 Objetivos

O objetivo deste trabalho foi de implementar um subconjunto da extensão bitmanip na arquitetura Ariane, um processador RISC-V disponibilizado recentemente e já em uso na academia e na indústria. Ariane é capaz de executar o sistema operacional Linux nos três níveis usuais de privilégio, o que o torna ideal para aplicações de uso geral. A ta-

refa proposta requeria, portanto, adaptar a arquitetura Ariane para decodificar e executar um número de instruções bitmanip. A adaptação deveria ser feita na descrição em linguagem de hardware (em System Verilog) da arquitetura. Havia alguma liberdade quanto à escolha das instruções particulares a se implementar, que deveriam ser apropriadas a uma primeira implementação experimental. Com esse fim, das 42 instruções disponíveis, seria implementado uma fração a ser escolhida durante o projeto. Ao fim deste, esperava-se obter um processador capaz de executar programas com as novas instruções bitmanip.

1.2 As circunstâncias do trabalho

Este trabalho foi realizado como estágio de conclusão do curso no laboratório Xmen, da Universidade Federal de Campina Grande. O estágio teve uma duração de 5 semanas.

1.3 A organização do relatório

Este documento é organizado como segue. Na 2ª seção, para fornecer o contexto técnico das adaptações de Ariane, é feita uma apresentação da arquitetura desse processador. Na 3º seção, introduz-se o projeto RISC-V Bitmanip e o subconjunto de instruções que se decidiu implementar. A 4ª seção trata das alterações realizadas no processador. A 5ª seção descreve a metodologia de validação e discute os resultados do trabalho.

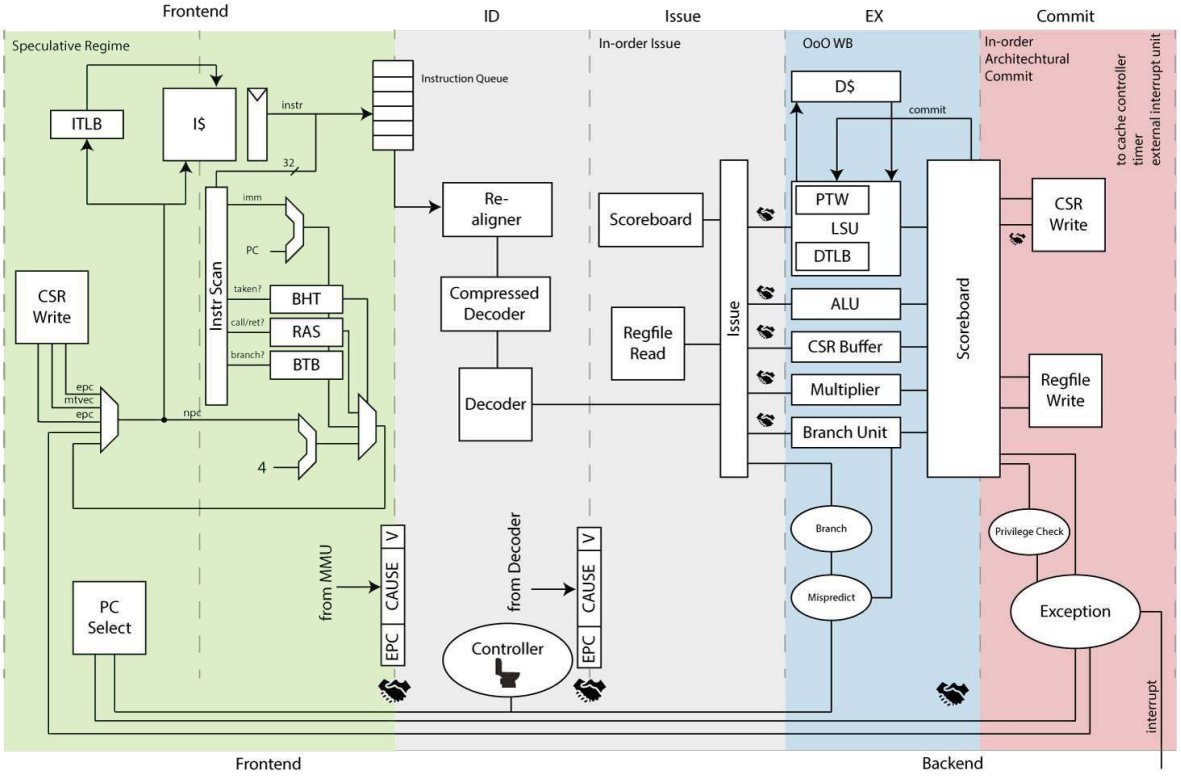
2 Ariane RISC-V

Ariane é uma das arquiteturas RISC-V desenvolvidas pelo grupo PULP, formado por uma parceria entre pesquisadores da escola ETH de Zurique e da Universidade de Bolonha. Disponibilizada em 2018 na plataforma Github, Ariane é o primeiro processador de "uso geral" desenvolvido pelo grupo: é capaz de executar Linux nos três níveis usuais de privilégio e implementa o conjunto RISC-V, 64 bits, com as extensões "I" (básica, de inteiros), "M" (multiplicação e divisão), "F" (ponto flutuante) e "A" (instruções atômicas) (ZARUBA, 2018b) (ZARUBA; BENINI, 2019).

O *pipeline* de Ariane tem seis estágios, como ilustrado na Figura 1. As funções de cada estágio se assemelham às encontradas tipicamente em processadores RISC (ZARUBA, 2018a):

- geração de PC (*program counter*): calcula o novo endereço virtual por simples incremento do valor anterior ou por aplicação de uma mudança de fluxo de execução, como em caso de *branches*;

Figura 1 – Pipeline do processador Ariane RISC-V.



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

- leitura de instrução (*instruction fetch*): traduz o endereço virtual em físico e realiza a leitura correspondente na *cache* de instruções;
- decodificação (*instruction decode*): produz as palavras de controle necessárias para executar a instrução recebida;
- leitura de operandos (*issue*): lê operandos nos bancos de registros inteiros ou flutuantes; detecta possíveis conflitos de dependência;
- execução (*execution*): processa a operação em uma das unidades funcionais;
- submissão de resultado (*commit*): autoriza a gravação do resultado no banco de registros ou na *cache* de dados.

No curso deste trabalho, o *pipeline* de Ariane foi adaptado para processar novas instruções *bitmanip*. Como as alterações mais extensas ocorreram nos estágios de leitura de operandos e de execução, estes serão descritos com mais detalhes nas subseções seguintes.

2.1 Leitura de operandos

O papel do estágio de leitura de operandos é fornecer operações a uma das unidades funcionais do estágio de execução. Além disso, ele armazena temporariamente o resultado da operação (*write-back*) e é responsável por comunicar ao estágio de *commit* que uma instrução foi finalizada. Essas funções são realizadas por meio de uma interação entre dois módulos: *issue and read operands* (IRO) e *scoreboard* (Figura 2).

2.1.1 *Scoreboard*

O *scoreboard* é uma das estruturas de controle fundamentais de Ariane. Consiste em uma fila FIFO ("primeiro a entrar, primeiro a sair") de instruções decodificadas, amparada por uma lógica combinatória para detecção de dependências (ZARUBA, 2018a). Quando o módulo IRO aceita uma nova instrução do decodificador, os campos correspondentes à instrução são escritos neste módulo. A estrutura de dados de uma entrada do *scoreboard* é descrita na Tabela 1.

O *scoreboard* serve a dois propósitos. Em primeiro lugar, ele armazena temporariamente o resultado de *write-back*, isto é, aquele proveniente do estágio de execução assim que a operação é concluída. Uma vez ela tenha sido finalizada e as instruções precedentes tenham sido submetidas, o *scoreboard* solicita ao estágio de *commit* que o resultado da nova instrução seja submetido – isto é, escrito no banco de registros ou na *cache* de dados.

Em segundo lugar, o *scoreboard* detecta dependências de leitura e escrita entre instruções e antecipa operandos quando necessário. Esse é o caso, por exemplo, quando

Tabela 1 – Estrutura de uma entrada do *scoreboard*

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

o operando de uma nova instrução deverá ser reescrito por uma instrução anterior que, no entanto, ainda não se concluiu. A detecção é possível porque o *scoreboard* mantém gravados os endereços de operandos e de resultado de cada instrução, como mostrado na Tabela 1.

Figura 2 – Estágio de leitura de operandos.

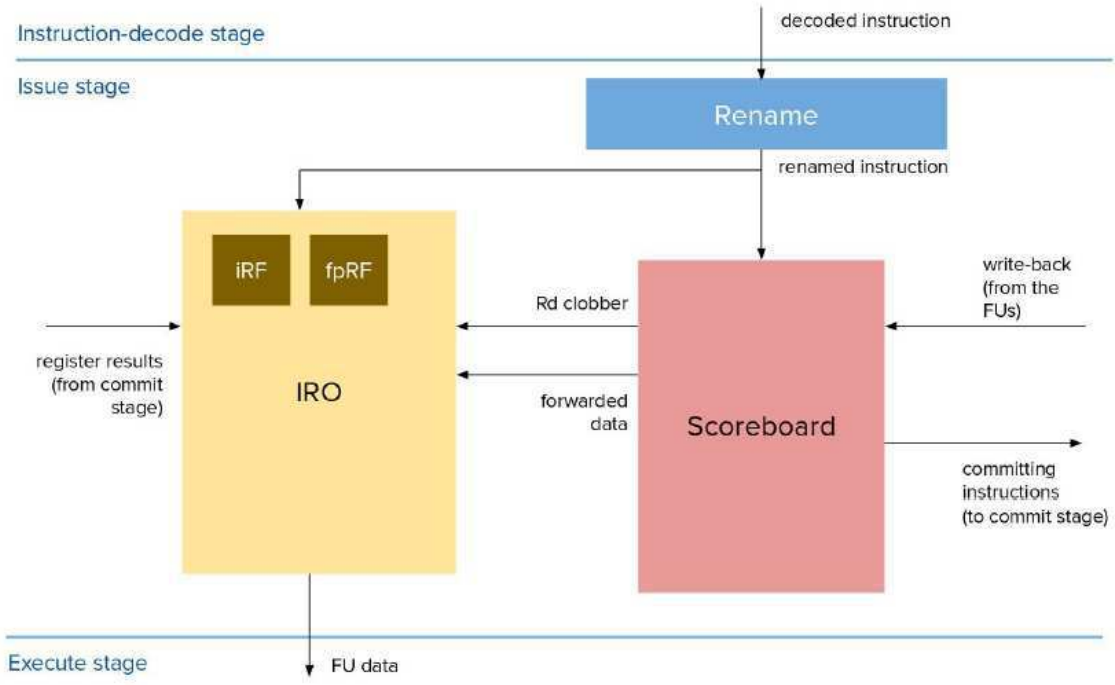


Tabela 2 – Interface do módulo *issue and read operands* com unidades funcionais

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

2.1.2 *Issue and read operands*

O módulo *issue and read operands* contém o banco de registros de inteiros e o de números flutuantes. Ele realiza a leitura de operandos em um dos registros e os encaminha a uma das unidades funcionais (UFs) do estágio de execução. Esse estágio se comunica com cada uma das UFs individualmente. A interface é apresentada na Tabela 2. Além dos operandos e dos sinais de *handshake*, nota-se também a presença de um "identificador de transação", que corresponde ao número de entrada da instrução na FIFO do scoreboard. Sua função é identificar a entrada nessa fila em que o resultado de *write-back* deverá ser escrito, uma vez que a operação tiver sido executada.

2.2 O estágio de execução

O estágio de execução de Ariane é composto de cinco unidades funcionais:

- unidade lógico-aritmética (ULA);
- unidade de *branch*;
- *buffer* de CSR, que contém registros de estado;
- unidade de ponto flutuante (FPU);
- unidade de *load* e *store* (LSU).

As unidades funcionais recebem operações do estágio anterior por meio da interface já descrita na Tabela 2. Ao fim da execução, elas comunicam o resultado e o ID de transação ao *scoreboard*, cada qual com uma porta em separado. A interface desses sinais de *write-back* é apresentada na Tabela 3.

Tabela 3 – Sinais de write-back enviados da UF ao scoreboard

Sinal	**Descrição**
Result (out)	Resultado da operação
Transaction ID (out)	Entrada da instrução no scoreboard
Valid (out)	Resultado é válido

3 A extensão RISC-V Bitmanip

A extensão RISC-V Bitmanip é desenvolvida por um dos grupos de trabalho da fundação RISC-V. O objetivo do grupo, que disponibiliza seus resultados na plataforma Github, é especificar um conjunto de instruções de manipulação de bits para processadores RISC-V. A seleção de instruções é orientada pelas seguintes diretrizes (WOLFF, 2019a) :

- Ganho em ciclos: em geral, cada nova instrução deve ser capaz de substituir três ou duas instruções básicas.
- Valor de desempenho: as instruções propostas devem ter utilidade em aplicações do mundo real. Essa utilidade será medida pelo desempenho em testes de benchmark.
- Simplicidade de hardware: as propostas não devem resultar em grande aumento de complexidade e área no processador.
- Consistência arquitetural: as propostas devem desviar o mínimo possível dos padrões do conjunto RISC-V, como quanto à codificação de instruções. Além disso, não devem implementar características já encontradas no conjunto existente.

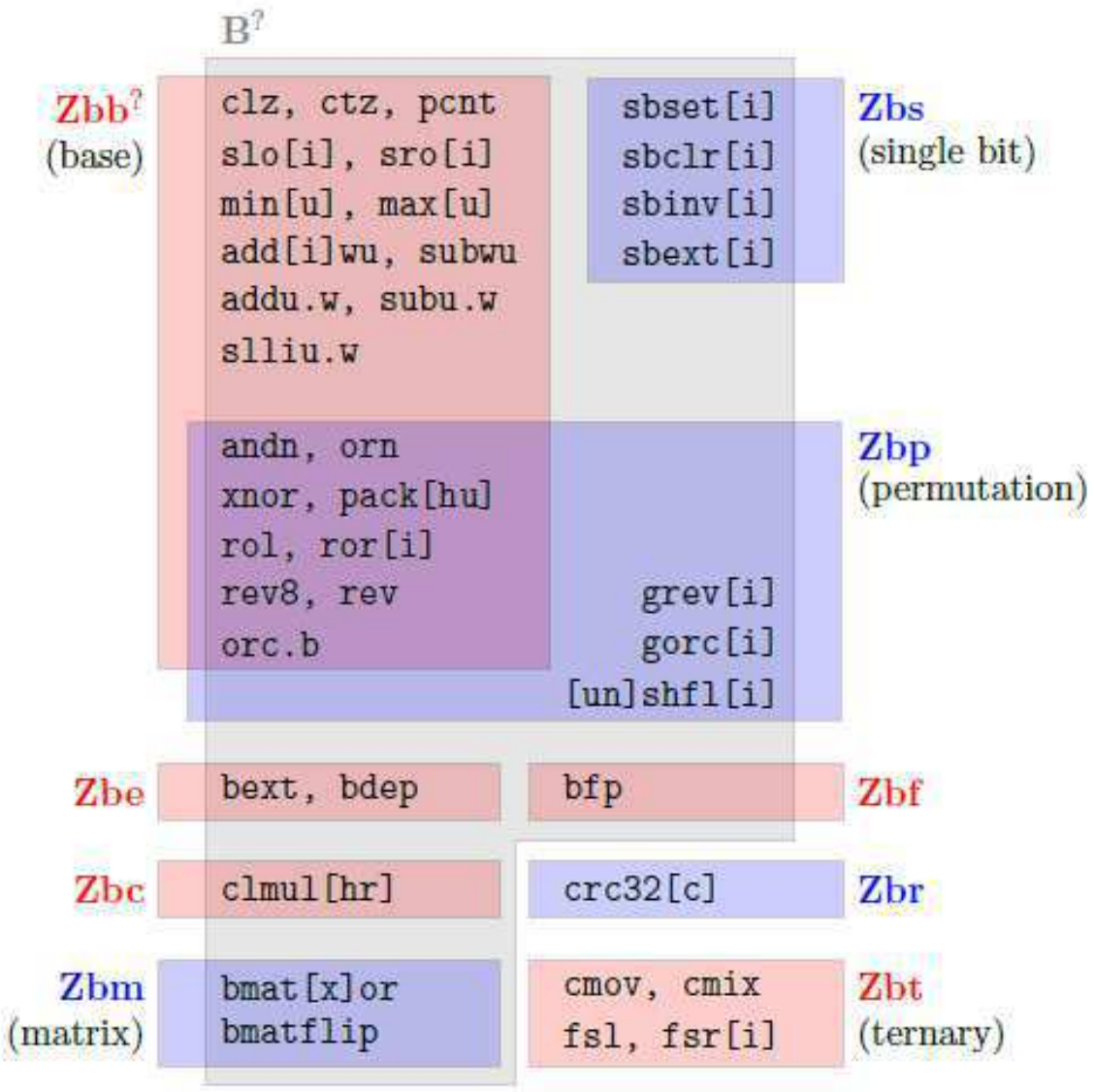
Embora a proposta ainda não tenha sido aprovada como extensão padrão do RISC-V, o projeto encontra-se em estado avançado de desenvolvimento. Um total de 42 instruções (com variantes) já têm seu comportamento especificado, embora a proposta de codificação em binário esteja sujeita a mudanças futuras. Esse conjunto é apresentado na Figura 3. Além disso, também são fornecidas ferramentas para teste em *software* e um grupo de implementações de referência em Verilog, (WOLFF, 2019b). Uma dessas implementações foi usada neste projeto.

3.1 O subconjunto implementado neste projeto

Do total de 42 instruções especificadas, foram escolhidas 6 para uma implementação inicial em Ariane. Todas elas são unárias – isto é, manipulam apenas um operando – e simples o suficiente para permitir execução em um ciclo. Instruções com essas características exigem um menor número de palavras de controle (pois há um só registro de operando) e um mecanismo de *handshake* mais simples (pois a unidade funcional estará sempre disponível), e portanto foram consideradas ideais para uma primeira implementação. Por outro lado, o trabalho foi planejado para que uma futura extensão ao conjunto completo possa ser feita sem grande aumento de complexidade.

As instruções escolhidas (descritas na Tabela 3) são implementadas por um dos módulos de referência fornecidos pelo grupo RISC-V Bitmanip. Por ser uma implementação já submetida a testes exaustivos, decidiu-se integrar esse mesmo módulo, com pequenas adaptações de interface, ao estágio de execução de Ariane.

Figura 3 – Proposta de extensão RISC-V bitmanip.



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

Tabela 4 – Instruções bitmanip implementadas

Instrução	**Descrição**
CLZ - Count Leading Zeroes	Conta número de zeros a partir do bit mais significativo
CTZ - Count Trailing Zeroes	Conta número de zeros a partir do bit menos significativo
PCNT - Population Count	Conta número de uns
SEXT.B - Sign Extend Byte	Estende sinal do byte menos significativo
SEXT.H - Sign Extend Hex	Estende sinal da palavra de dois bytes menos significativa
BMATFLIP - Byte Matrix Flip	Trata operando de 64 bits como matriz de 8x8 bytes. Produz a matriz transposta

4 Adaptação da arquitetura

4.1 Inserção da unidade *bitmanip* no estágio de execução

A interface do módulo de referência, que será denominado contador de bits, é apresentada na Tabela 5. Para adequá-lo à interface do estágio de execução de Ariane, esse módulo foi inserido em um envelope (*wrapper*) com interface que se assemelha à das demais unidades funcionais. A interface do módulo de envelope, que se chamou unidade *bitmanip*, é descrita na Tabela 6.

Em Ariane, cada operação é especificada por um dos valores do tipo *fu_op*, de 7 bits. Assim, para tornar as novas operações reconhecíveis, 6 novos valores foram acrescentados a esse tipo, correspondendo às 6 instruções *bitmanip*. Além disso, um novo valor *BITMANIP* foi acrescentado ao tipo *fu_t*, que especifica a UF a ser usada.

Na interface do contador de bits, nota-se a que operação é codificada por um conjunto de 4 bits. Para passagem da codificação de Ariane para a do contador, um simples bloco de conversão foi inserido no envelope.

4.2 Mudanças no estágio de leitura de operandos

O estágio de leitura de operandos sofreu mudanças nos módulos *issue and read operands* e *scoreboard*. No primeiro, foi acrescentada a interface para transferência de operandos à unidade *bitmanip* – isto é, os sinais *FU_data* e *bitmanip_valid_i* da Tabela

Tabela 5 – Interface do módulo contador de bits

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

Tabela 6 – Interface da unidade de bitmanip, envelope do contador de bits

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

6. Além disso, a lógica do protocolo de *handshake* foi estendida para levar em conta o sinal de *ready* da nova unidade. Como o contador de bits é um módulo combinatório, esse sinal estará sempre ativo. Ele passará a ser relevante, no entanto, quando outros módulos de referência (alguns dos quais são sequenciais) forem incluídos no envelope.

No scoreboard, foram acrescentadas portas para os sinais de *write-back* da nova unidade (Tabela 3). Devido à modularidade de design de Ariane, isso pôde ser feito com um simples incremento de parâmetro. A lógica para detecção de dependências não precisou ser modificada.

4.3 Mudanças no decodificador

A codificação das 6 novas instruções bitmanip é apresentada na Figura 4. O grupo RISC-V Bitmanip decidiu implementá-las com um opcode já usado para operações com valor imediato, como é o caso de instruções lógicas (*and*, *or*, *xor* etc) e de deslocamento (*sll*, *srl*, *sra* etc), por exemplo. A triagem das instruções dentro desse grupo é feita a partir dos bits 31-25 (conhecidos como *func7*) e 24-20. Estes últimos são usados para codificar o segundo operando (*rs2*) em outras instruções (WOLFF, 2019a).

A lógica para identificação das novas instruções foi incluída no decodificador de Ariane, com geração das palavras de controle correspondentes. Como todas as instruções são unárias, foi preciso identificar apenas a unidade funcional (i.e., a unidade *bitmanip*), a operação, o endereço do primeiro operando (*rs1*) e o do registro de destino (*rd*).

As mudanças descritas nesta seção constituem toda a adaptação realizada em Ariane para incluir as novas instruções *bitmanip*. Na seção seguinte, é descrita a metodologia

de validação da arquitetura.

Figura 4 – Codificação das instruções bitmanip implementadas.

3	2	1		
1 0 9 8 7 6 5 4 3 2 1 0 9 8 7 6 5 4 3 2 1 0 9 8 7 6 5 4 3 2 1 0				
0110000	00000	rs1	001	rd	0010011	CLZ
0110000	00001	rs1	001	rd	0010011	CTZ
0110000	00010	rs1	001	rd	0010011	PCNT
0110000	00011	rs1	001	rd	0010011	BMATFLIP
0110000	00100	rs1	001	rd	0010011	SEXT.B
0110000	00101	rs1	001	rd	0010011	SEXT.H

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

5 Validação e resultados

Como dito acima, o módulo contador de bits é dado como referência no projeto RISC-V Bitmanip e já foi submetido aos testes padrão de seus autores. A metodologia de testes desse módulo consiste em estimulá-lo com uma série de valores de operandos e, para cada uma das 6 operações suportadas, comparar os resultados obtidos com valores de referência, que são calculados a partir dos modelos em C. Como esse teste é praticamente exaustivo, considerou-se que o módulo contador de bits estava devidamente validado. Restava testar o envelope do módulo e as alterações feitas em Ariane.

Assim, o objetivo dos testes neste trabalho foi de verificar que Ariane é capaz de decodificar as novas instruções e fornecer os operandos corretos à unidade *bitmanip*, assim como de escrever o resultado no registrador de destino correto. Como isso requer um conhecimento detalhado das trocas de sinais entre as partes de Ariane, essa análise foi feita por estudo das formas de onda em um *software* de simulação. As ferramentas escolhidas para compilação, simulação e visualização foram Xcelium, Incisive e SimVision, do grupo Cadence.

5.1 Metodologia de teste e resultados

Foram escritos programas curtos em *assembly* RISC-V para testar cada nova instrução. As novas instruções foram implementadas como macros de *assembly*. Assim, por exemplo, a expressão “clz rd, rs1” é traduzida em uma palavra de 32 bits contendo o *opcode* de *clz* e o número binário dos registros *rd* e *rs1*.

O inconveniente desse método é que, como macros de *assembly* admitem pouca flexibilidade, os números dos registros deviam ser escritos diretamente em sua forma binária – isto é, por exemplo: “clz 00100, 00000”. Uma estratégia melhor teria sido a de incluir as novas instruções no próprio compilador RISC-V, ou de explorar opções já oferecidas pelo grupo RISC-V Bitmanip. Pela simplicidade de implementação, preferiu-se o método descrito acima. A Tabela 7 apresenta as macros de *assembly* empregadas.

Usou-se como plataforma de testes um módulo em System Verilog já fornecido pelos desenvolvedores de Ariane. Esse módulo, que se chama *test harness*, contém uma instância de Ariane, uma memória ROM para boot e uma SRAM como memória principal, além de periféricos como porta UART. O *testbench* desse módulo, também fornecido com Ariane, foi adaptado para as ferramentas disponíveis no laboratório. Após os primeiros ciclos de simulação, o programa que se deseja testar é carregado na memória principal e passa a ser lido pelo processador.

Tabela 7 – Implementação das instruções bitmanip como macros em assembly

```
.macro clz rd, rs, func3=001, opcode=0010011
.word 0b011000000000\rs\func3\rd\opcode
.endm

.macro ctz rd, rs, func3=001, opcode=0010011
.word 0b011000000001\rs\func3\rd\opcode
.endm

.macro pcnt rd, rs, func3=001, opcode=0010011
.word 0b011000000010\rs\func3\rd\opcode
.endm

.macro bmatflip rd, rs, func3=001, opcode=0010011
.word 0b011000000011\rs\func3\rd\opcode
.endm

.macro sext.b rd, rs, func3=001, opcode=0010011
.word 0b011000000100\rs\func3\rd\opcode
.endm

.macro sext.h rd, rs, func3=001, opcode=0010011
.word 0b011000000101\rs\func3\rd\opcode
.endm
```

Como explicado acima, o objetivo dos testes foi verificar que Ariane é capaz de decodificar, executar e submeter resultados das instruções bitmanip. Essa verificação foi feita por uma análise das formas de onda do *pipeline* (figuras 5 e 6). Especificamente, desejou-se verificar que:

- o decodificador produz as palavras de controle esperadas;
- o estágio de leitura de operandos lê o registro correto e atribui a operação à unidade de *bitmanip*;
- essa unidade recebe o operando, o código de operação e o ID de transação corretos;
- o resultado esperado é escrito no banco de registros inteiros;
- o registro de destino é aquele especificado na instrução;
- as etapas acima se processam no número de ciclos esperado; isto é, por exemplo, a unidade de bitmanip produz um resultado no mesmo ciclo em que recebe a operação.

Além disso, foi feito um teste de conflitos de dependência dos tipos “leitura após escrita” (RAW, em inglês) e “escrita após leitura” (WAR) entre instruções bitmanip e

Tabela 8 – Programas de teste

Programa	**Características testadas**
bitcnt_test.s	Instruções clz, ctz, pcnt, sext.b e sext.h
bmaflip.s	Instrução bmatflip
ctz_test.s	Instrução ctz
dhazards.s	Dependência de dados de tipos RAW e WAR

instruções convencionais. O objetivo era verificar que o scoreboard paralisa a operação ou antecipa os operandos corretamente. A Tabela 8 descreve os programas de teste.

Além das formas de onda, a análise da execução foi simplificada pelo uso de uma ferramenta já fornecida pela equipe de Ariane. Um módulo "marcador de instruções" (*instruction tracer*) sonda sinais do *pipeline* para detectar qual instrução foi submetida (*committed*) em certo ciclo, além de identificar seus operandos e seu registro de destino (Figura 7). Esse módulo permite reconhecer o caso em que uma instrução *bitmanip* é submetida, o que significa ao menos que o *pipeline* não foi paralisado. No entanto, o método não permite ver o resultado da operação ou a causa do problema quando o *pipeline* é parado por um conflito de *handshake*, por exemplo.

Com base nos testes acima descritos, pôde-se concluir que a arquitetura adaptada se comporta como esperado quando da execução das seis novas instruções. Isto é, sua decodificação, sua execução e a submissão de resultados são feitos corretamente, e conflitos

Figura 5 – Formas de onda no decoder: decodificação de quatro instruções bitmanip

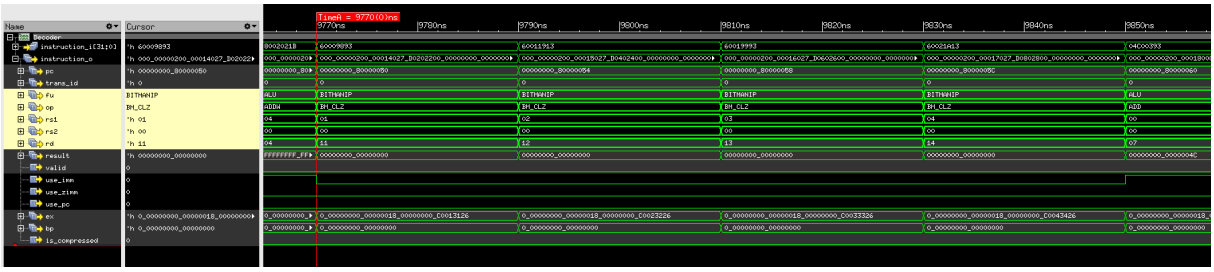
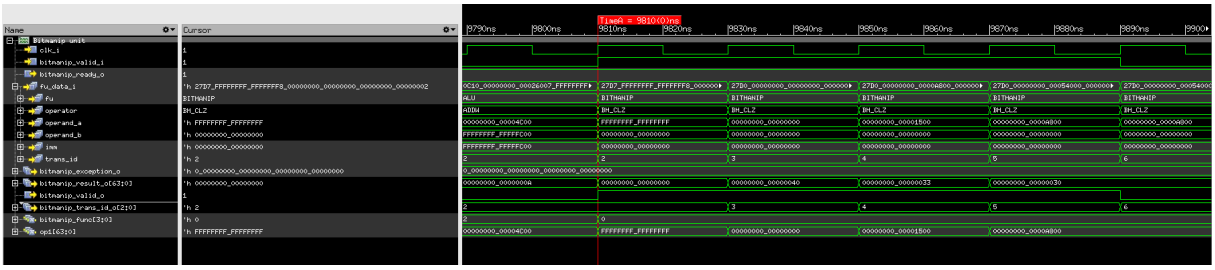


Figura 6 – Formas de onda na unidade de bitmanip: recepção de quatro operações



de dependência do tipo RAW e WAR são contornados. Uma validação mais criteriosa exigiria, porém, certos tipos de teste que não foram possíveis devido à limitação do tempo de estágio. Na subseção seguinte, são feitas algumas sugestões para aprimorar a validação da arquitetura em trabalhos futuros.

5.2 Limitações dos testes realizados e sugestões para melhoria

Em primeiro lugar, uma validação rigorosa exigiria testes de não regressão – isto é, testes da arquitetura modificada para as operações já suportadas pela arquitetura original. Isso pode ser feito com uma aplicação do conjunto de testes torture para RISC-V, que é disponibilizado pela equipe de Ariane. A aplicação dos testes exige uma adaptação do arquivo *make* às ferramentas de simulação do laboratório.

Foram realizados testes de dependência dos tipos RAW ("leitura após escrita") e WAR ("escrita após leitura") entre instruções bitmanip e uma instrução convencional, em ambas as sequências bitmanip-convencional e convencional-bitmanip. Um exame do mecanismo de detecção de dependências em Ariane permitiria afirmar que esses testes são suficientes. Uma validação mais rigorosa, porém, exigiria testes na sequência bitmanip-bitmanip (para cada instrução do tipo) e entre instruções bitmanip e instruções convencionais não testadas, como as de load e store.

Como defendido em seção anterior, o módulo contador de bits, fornecido pelo grupo

Figura 7 – Texto de saída do módulo instruction tracer.

```
xcelium> cat trace_hart_0.log
    5510ns       268 M 0000000000010000 0 0010041b addiw          s0, 1                s0  :0000000000000001
    5550ns       270 M 0000000000010004 0 01f41413 slli           s0, s0, 0x1f         s0  :0000000080000000 s0  :0000000000000001
    5570ns       271 M 0000000000010008 0 f1402573 csrr           a0, mhartid          a0  :0000000000000000
    5610ns       273 M 000000000001000c 0 00000597 auipc          a1, 0x0              a1  :000000000001000c
    5690ns       277 M 0000000000010010 0 07458593 addi           a1, a1, 116          a1  :000000000001008 0 a1  :000000000001000c
    5710ns       278 M 0000000000010014 1 00008402 c.jr           x0, s0, 0            s0  :0000000080000000
    5990ns       292 M 0000000080000000 0 00000013 nop
    6010ns       293 M 0000000080000004 0 00000013 nop
    6030ns       294 M 0000000080000008 0 00000013 nop
    6050ns       295 M 000000008000000c 0 00000013 nop
    6450ns       315 M 0000000080000010 0 00000013 nop
    6470ns       316 M 0000000080000014 0 00000013 nop
    6490ns       317 M 0000000080000018 0 00000013 nop
    6510ns       318 M 000000008000001c 0 fff00093 li             ra, -1               ra  :ffffffffffffffff
    7230ns       354 M 0000000080000020 0 00000113 li             sp, 0                sp  :0000000000000000
    7250ns       355 M 0000000080000024 0 000011b7 lui            gp, 0x1              gp  :0000000000001000
    7290ns       357 M 0000000080000028 0 5001819b addiw          gp, gp, 1280         gp  :0000000000001500 gp  :0000000000001000
    7310ns       358 M 000000008000002c 0 0000b237 lui            tp, 0xb              tp  :000000000000b000
    7850ns       385 M 0000000080000030 0 8002021b addiw          tp, tp, -2048        tp  :000000000000a800 tp  :000000000000b000
    7870ns       386 M 0000000080000034 0 fff0029b addiw          t0, -1               t0  :ffffffffffffffff
    7910ns       388 M 0000000080000038 0 03029293 slli           t0, t0, 0x30         t0  :ffff000000000000 t0  :ffffffffffffffff
    7930ns       389 M 000000008000003c 0 0c000313 li             t1, 192              t1  :00000000000000c0
    8970ns       441 M 0000000080000040 0 04c00393 li             t2, 76               t2  :000000000000004c
    8990ns       442 M 0000000080000044 0 0000c437 lui            s0, 0xc              s0  :000000000000c000
    9010ns       443 M 0000000080000048 0 000054b7 lui            s1, 0x5              s1  :0000000000005000
    9050ns       445 M 000000008000004c 0 c004849b addiw          s1, s1, -1024        s1  :0000000000004c00 s1  :0000000000005000
    9850ns       485 M 0000000080000050 0 60009893 INVALID
    9870ns       486 M 0000000080000054 0 60011913 INVALID
    9890ns       487 M 0000000080000058 0 60019993 INVALID
    9910ns       488 M 000000008000005c 0 60021a13 INVALID
   10130ns       499 M 0000000080000060 0 60029a93 INVALID
   10150ns       500 M 0000000080000064 0 60109893 INVALID
   10170ns       501 M 0000000080000068 0 60111913 INVALID
   10190ns       502 M 000000008000006c 0 60119993 INVALID
   10590ns       522 M 0000000080000070 0 60121a13 INVALID
   10610ns       523 M 0000000080000074 0 60129a93 INVALID
   10630ns       524 M 0000000080000078 0 60209893 INVALID
   10650ns       525 M 000000008000007c 0 60211913 INVALID
   11430ns       564 M 0000000080000080 0 60219993 INVALID
   11450ns       565 M 0000000080000084 0 60221a13 INVALID
   11470ns       566 M 0000000080000088 0 60229a93 INVALID
   11490ns       567 M 000000008000008c 0 60431b13 INVALID
   12030ns       594 M 0000000080000090 0 60439b93 INVALID
   12050ns       595 M 0000000080000094 0 60541c13 INVALID
   12070ns       596 M 0000000080000098 0 60549c93 INVALID
   12090ns       597 M 000000008000009c 0 0000006f j              pc - 0
   12150ns       600 M 000000008000009c 0 0000006f j              pc - 0
   12210ns       603 M 000000008000009c 0 0000006f j              pc - 0
   12270ns       606 M 000000008000009c 0 0000006f j              pc - 0
   12330ns       609 M 000000008000009c 0 0000006f j              pc - 0
```

Notam-se instruções bitmanip marcadas como "invalid".

RISC-V Bitmanip, foi considerado como devidamente validado por seus autores. Isto é, considerou-se que cada uma das seis operações é executada corretamente para qualquer valor de operando. Inserido esse módulo no estágio de execução de Ariane, restou verificar que o processador é capaz de lhe fornecer operandos e gravar resultados corretamente, bem como que o envelope é transparente a esses operandos e ao resultado do contador. Assim, notou-se que os resultados produzidos pelo contador são os esperados para todos os operandos fornecidos, mas não se procurou testá-lo exaustivamente para vários valores de operando. Uma validação rigorosa talvez exija aplicação de testes exaustivos. Isso pode ser feito com uma adaptação do testbench do contador ao contexto de Ariane.

Por fim, o método de leitura dos resultados pode ser alterado para permitir uma automatização do procedimento. Como dito acima, a validação foi feita por exame das formas de onda em um software de simulação. Se for permitido diminuir seu nível de detalhes, ela poderá ser feita por impressão dos resultados através da porta UART, com posterior leitura do arquivo de transcrição por um programa.

6 Conclusão

Neste trabalho, foi descrita a adaptação da arquitetura Ariane a um subconjunto da extensão RISC-V bitmanip.

Das 42 instruções especificadas pelo grupo RISC-V Bitmanip, foram escolhidas 6 para uma implementação inicial em Ariane. Fez-se uso de um módulo de referência já disponibilizado pelo grupo para execução das 6 operações. Esse módulo foi inserido em um envelope para adequação de sua interface à usada em Ariane, e em seguida inserido no estágio de execução do processador. A arquitetura foi então adaptada para processar as 6 instruções *bitmanip* acrescentadas.

A arquitetura foi validada por execução de programas de teste e por análise das formas de onda em um *software* de simulação. Os testes visaram verificar o processamento individual de cada nova instrução e também sua interação com instruções convencionais por conflitos de dependência de operando. São feitas sugestões para uma validação mais exaustiva no futuro, como por realização de testes de não regressão, testes de dependência adicionais e testes exaustivos das operações *bitmanip*.

Bibliografia

WOLFF, C. *Bitmanip RISC-V Extension*. [S.l.]: Acesso: `https://github.com/riscv/riscv-bitmanip/bitmanip-0.92.pdf`, 2019. Citado 3 vezes nas páginas 9, 16 e 20.

WOLFF, C. *RISC-V Bitmanip (Bit Manipulation) Extension*. [S.l.]: Acesso: `https://github.com/riscv/riscv-bitmanip`, 2019. Citado na página 16.

ZARUBA, F. *Ariane*. [S.l.]: Acesso: `https://pulp-platform.github.io/ariane/docs/home/`, 2018. Citado 2 vezes nas páginas 11 e 12.

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.