VIII Simpósio de Iniciação Científica 2025

"Do conhecimento acadêmico à transformação sustentável: inovação com validação científica"

SÍNTESE DE MICROARQUITETURA RISC-V RV32IM.

José Augusto dos Santos Lemos (IC), Gustavo Della Colletta (PQ)1
[1]*Universidade Federal de Itajubá.*

Palavras-chave: Arquitetura e Organização de Computadores. Computação de Alto Desempenho. RISC-V.

Introdução

O conjunto de instruções RISC-V é gratuito e aberto, desenvolvido na Universidade da Califórnia em Berkeley, designado ao desenvolvimento do hardware open-source no contexto da arquitetura e organização de computadores. O RISC- V possui um subconjunto base de instruções "I" e extensões opcionais que adicionam outras instruções mais complexas, para processadores de 32, 64 e 128 bits, dessa forma, o hardware pode ser projetado de acordo com a necessidade da aplicação (PATTERSON & WATERMAN, 2019). O objetivo deste trabalho, é implementar uma microarquitetura de um processador RISC-V de 32 bits com as instruções do conjunto base "I" e com as instruções de multiplicação e divisão de números inteiros presentes na extensão "M", e otimizar a área e o desempenho do processador. Essa implementação desconsidera as instruções privilegiadas, o tratamento de exceções ou alterações no modo de operação da microarquitetura.

Metodologia

O processador RV32IM utiliza a microarquitetura pipeline de 5 estágios e foi descrita em linguagem Verilog. Dentre as vantagens da microarquitetura pipeline, cabe destacar maiores frequências de operação e throughput (HENNESSY & PATTERSON, 2020). Sua implementação e síntese foram realizadas no toolchain Vivado Design Suite 24.2, da Xilinx, tendo como alvo a FPGA Artix-7 presente na placa de desenvolvimento Nexys A7-50T da Digilent. Para este dispositivo e utilizando estas ferramentas, foram analisados a latência, a máxima frequência de clock, e o consumo de elementos lógicos para o processador. Com o objetivo de otimizar o desempenho, foi adotada uma estratégia de projeto assimétrica para as unidades de multiplicação e divisão. O circuito multiplicador foi projetado para baixa latência, minimizando seu impacto na frequência de clock, dado que as instruções de multiplicação são muito frequentes. Em contrapartida, o circuito divisor foi otimizado para um baixo consumo de recursos lógicos, uma vez que as instruções de divisão são raramente utilizadas em programas de larga escala (HENNESSY, 2019). O principal gargalo de desempenho em circuitos multiplicadores reside na necessidade de somar um grande número de produtos parciais. Essa característica resulta, em abordagens tradicionais, em alta latência e elevado consumo de recursos lógicos. Para contornar essa limitação, optou-se por uma arquitetura composta que combina o algoritmo de Booth modificado com a topologia *Wallace Tree* e um somador de prefixos paralelos como somador final *(SAHOO; MAHAPATRA, 2015)*. O algoritmo modificado de Booth é utilizado para reduzir pela metade a quantidade de produtos parciais, na qual o algoritmo encontra redundâncias no processo de multiplicação, por meio de longas sequências de 1s ou 0s, diminuindo diretamente a complexidade do circuito e a área ocupada. Subsequentemente, a topologia *Wallace Tree* realiza a soma desses produtos parciais de forma paralela e altamente eficiente, utilizando uma estrutura de somadores do tipo *Carry Save* ou CSAs que reduz a profundidade lógica do caminho crítico, devido ao funcionamento dos somadores CSAs atrasam a propagação dos *Carries* para a próxima camada, garantindo uma alta velocidade. A etapa final é executada por um somador de alta velocidade, em que foi escolhido um somador de prefixo paralelo. A combinação destas técnicas resulta em um multiplicador com uma relação otimizada entre velocidade e área. Para o multiplicador de 32 bits, o codificador de *Booth*, que é o circuito combinacional que implementa o algoritmo modificado de *Booth*, reduz a quantidade de produtos parciais pela metade, na qual são produzidos 16 produtos parciais de 64 e bits. Os somadores *Carry Save* ou CSAs, que podem ser interpretados como um compressor 3:2 devido ao fato de que recebe 3 entradas de N bits e produz 2 saídas também de N bits, são utilizados de modo a realizar a soma desses produtos parciais de forma paralela e eficiente no multiplicador *Wallace Tree*, em que a árvore de soma comprime os 16

produtos parciais de 64 bits em apenas 2 produtos parciais de 64 bits que precisam ser somados por um somador final. A topologia dos somadores CSA, é mostrada na figura 1, o multiplicador *Wallace Tree* é mostrada na figura 2 e o multiplicador *Wallace Tree* com algoritmo modificado de *Booth* é mostrado na figura 3.

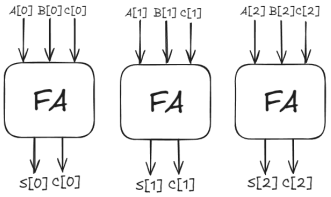

Figura 1 – Somador Carry Save.

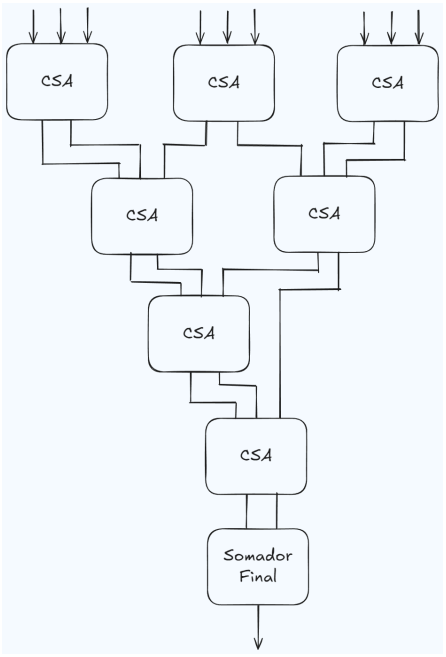

Figura 2 – Multiplicador *Wallace Tree*.

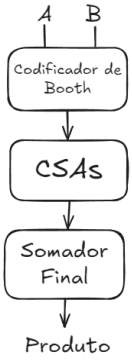

Figura 3 – Multiplicador *Wallace Tree* com algoritmo modificado de *Booth*.

O somador de prefixos paralelos utilizado no circuito multiplicador como somador final é o somador de *Knowles*, sendo uma topologia de alta velocidade que realiza o cálculo dos *Carries* de forma paralela e eficiente, similar ao somador *Kogge Stone*, porém com um consumo de área levemente menor(TALSANIA; JOHN, 2012). A figura 4 mostra a topologia do somador de *Knowles* de 16 bits em que as caixas pretas são as células que produzem os prefixos *generate* e *propagate*, as células cinzas produzem apenas os prefixos *propagate* e os triângulos ou *buffers* não produzem nenhum prefixo, apenas propagam os prefixos recebidos.

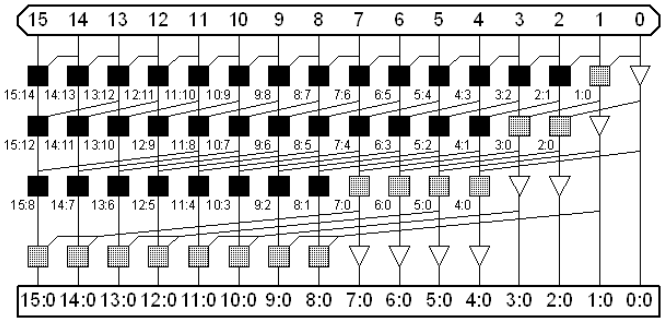

Figura 4 - Somador de *Knowles* de 16 bits.

O circuito divisor, responsável pela execução das instruções de divisão, foi projetado com o objetivo de operar em altas frequências de *Clock* e com um baixo consumo de elementos lógicos, em troca de uma alta latência . Esta abordagem se deve ao fato de que as instruções de divisão são raramente utilizadas em programas de larga escala, portanto, não há necessidade de um divisor de alto desempenho.O algoritmo de divisão por restauração consiste em um método sequencial amplamente utilizado em implementações de hardware digital para operações de divisão. Seu funcionamento baseia-se no deslocamento sucessivo do dividendo parcial à esquerda, seguido da tentativa de subtração do divisor. Caso o resultado dessa operação seja maior ou igual a zero, a subtração é mantida e o bit correspondente do quociente assume valor igual a um. Por outro lado, se a operação resultar em valor negativo, o dividendo parcial é restaurado ao seu valor anterior, por meio da adição do divisor, e o bit do quociente é definido como zero. Esse procedimento é repetido iterativamente até que todos os bits do quociente sejam determinados, resultando ao final no valor do quociente e do resto da divisão. A implementação deste algoritmo em hardware, resulta em um circuito sequencial, em que

neste caso para valores de 32 bits, o circuito requer 32 ciclos de *Clock* para completar a operação e portanto, as instruções de divisão são instruções multiciclo que requerem 32 ciclos para serem executadas. A topologia utilizada para o circuito divisor é a do algoritmo de restauração, mostrado na figura 5.

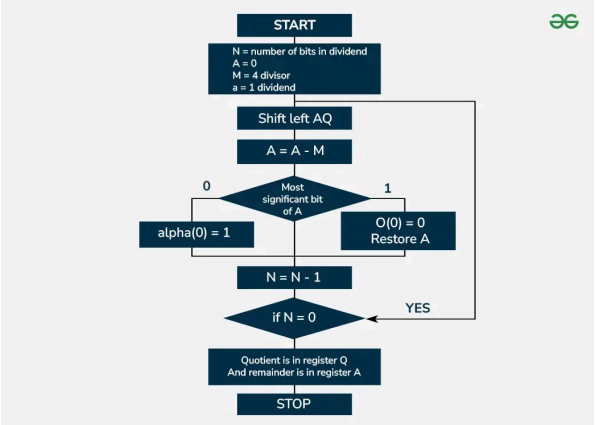

Figura 5 - Fluxograma do algoritmo de divisão por restauração.

Resultados e discussão

O processador RISC-V RV32IM com o multiplicador *Wallace Tree* e o circuito divisor, foi comparado a outra topologia com apenas as instruções de multiplicação, utilizando o multiplicador de Vedic (LEMOS & COLLETTA, 2024) em termos de elementos lógicos e frequência de *Clock*. Adicionalmente, para diminuir o caminho crítico e maximizar a frequência de operação, a própria estrutura do multiplicador foi segmentada com o uso de *pipeline*. Essa abordagem é particularmente eficiente no Wallace Tree, pois a árvore de soma com CSAs é consideravelmente mais rápida que o somador final. Os resultados para ambos os processadores estão presentes na tabela 1.

Tabela 1: Resultados dos processadores RISC-V.

Estrutura	Frequência	Elementos Lógicos
Vedic	41,89 MHz	3180
Wallace Tree e Divisor	74.1 MHz	3289

Percebe-se por estes resultados, que a topologia híbrida do multiplicador *Wallace Tree* com o algoritmo modificado de *Booth* e o somador de *Knowles*, proporciona um desempenho muito superior se comparado ao multiplicador de Vedic, com um custo de área levemente maior. De modo a testar a funcionalidade das instruções de divisão e multiplicação do processador RISC-V RV32IM, foi elaborado um código de testes e realizado uma simulação através da ferramenta Vivado Design Suite 2024.2, em que o código está presente na figura 6.

```
.globl _boot
.text

_boot:
   li x1, 20   # Unsigned Dividend
   li x2, 3    # Unsigned Divisor
   li x3, -20  # Signed Dividend
   li x4, -3   # Signed Divisor

   li x5, 6    # Quotient (20/3)
   li x6, 2    # Remainder (20%3)
   li x11, -2  # Remainder (-20%-3)

   # Unsigned division and remainder
   divu x7, x1, x2   # (20/3)
   remu x8, x1, x2   # (20%3)

   # Signed division and remainder
   div x9, x3, x4 # (-20/-3)
   rem x10, x3, x4   # (-20%-3)

   # Dividend = Divisor * Quotient + Remainder
   # Testing unsigned division
   mul x11, x2, x7   # Divisor * Quotient
   add x11, x11, x6 # Divisor * Quotient + Remainder

   # Comparing the result to the dividend
   bne x11, x1, fail

   # Testing signed division
   mul x12, x4, x9   # Divisor * Quotient
   add x12, x12,x10 # Divisor * Quotient + Remainder

   # Comparing the result to the dividend
   bne x12, x3, fail

   beq x0, x0, _boot

fail:
   beq x0, x0, fail
```

Figura 6 - Código de testes para o processador RISC-V.

Este programa testa as operações de divisão e resto para números inteiros, com e sem sinal. Para validar os resultados, ele confirma a identidade matemática Dividendo = (Divisor × Quociente) + Resto. Se todos os cálculos estiverem corretos, o programa entra em um laço de sucesso e caso contrário, desvia para um laço de falha. O programa de testes necessita de 87 ciclos de *Clock* do processador para ser executado, na qual a

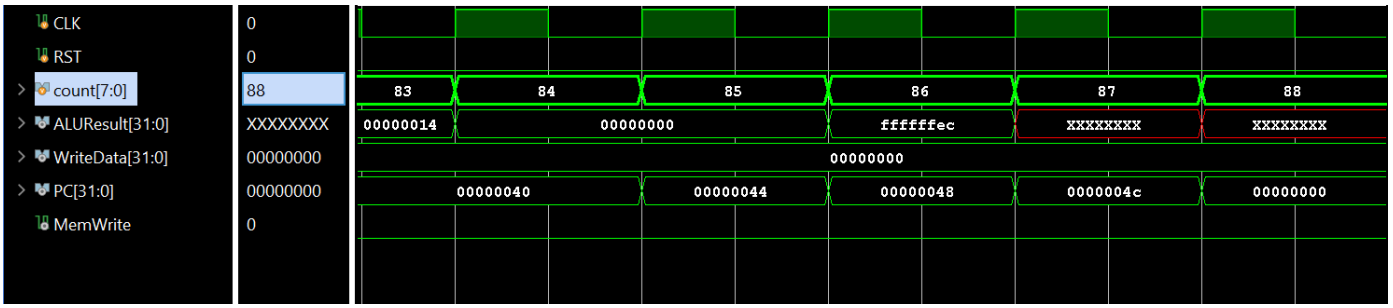

Figura 7 - Simulação das instruções de multiplicação e divisão.

verificação de sucesso para o programa consiste em determinar se o registro *Program Counter* ou PC, que indica o endereço da instrução a ser executada, retorna ao valor 0, no ciclo de *Clock* de número 88. Os resultados obtidos para a simulação estão presentes na figura 7, em que é possível ver que o programa retorna a primeira instrução no ciclo 88.

Conclusões

Os resultados indicam que de modo a atingir um alto nível de desempenho em processadores RISC-V com as instruções de multiplicação e divisão incluídas, é necessário a utilização de topologias de circuitos mais elaborados para a multiplicação, que resultam em um consumo maior de elementos lógicos, porém para as instruções de divisão, devido ao fato de serem pouco utilizadas em programas de larga escala, as topologias simples que proporcionam um menor consumo de elementos lógicos são mais eficientes para a aplicação. Os resultados também indicam a alta eficiência da topologia do multiplicador *Wallace Tree* em conjunto ao algoritmo modificado de *Booth*, na qual o processador com o multiplicador *Wallace Tree* com algoritmo modificado de *Booth* apresenta uma frequência de *Clock* significativamente maior se comparado ao processador com multiplicador de Vedic, com um consumo de elementos lógicos levemente maior.

Agradecimentos

Agradeço a Unifei e ao CNPQ pela oportunidade de aprofundar meus conhecimentos na área da eletrônica digital e arquitetura de computadores através desta pesquisa, de modo a melhorar meu desempenho acadêmico e profissional.

Referências

1. LEMOS, José Augusto dos Santos; COLLETTA, Gustavo Della. Multiplicadores binários em microarquitetura RISC-V. In: SIMPÓSIO DE INICIAÇÃO CIENTÍFICA DA UNIVERSIDADE FEDERAL DE ITAJUBÁ, 7., 2024, Itajubá.

2. WATERMAN, A.; ASANOVIC, K. The RISC-V Instruction Set Manual Volume I: Unprivileged.sty. [S.l.], 2019. Disponível em: <https://github.com/riscv/riscv-isa-manual/releases/download/Ratified-IMAFDQC/riscv-spec-20191213.pdf>.

3. HENNESSY, John L.; PATTERSON, David A. **Computer Architecture: A Hardware/Software Interface - RISC-V Edition.** 1. ed. Cambridge, MA: Morgan Kaufmann, 2020.

4. HENNESSY, John L.; PATTERSON, David A. *Arquitetura de Computadores: Uma Abordagem Quantitativa*. 6. ed. Rio de Janeiro: Grupo GEN, 2018.

5. TALSANIA, Megha; JOHN, Eugene. A Comparative Analysis of Parallel Prefix Adders.

6. SAHOO, S. K.; MAHAPATRA, K. K. Design of a High-Speed Hybrid Multiplier Using Booth Encoded Wallace Tree and Parallel Prefix Adder. In: INTERNATIONAL CONFERENCE ON COMMUNICATION, INFORMATION & COMPUTING TECHNOLOGY (ICCICT), 2015, Mumbai. **Anais...** Mumbai: IEEE, 2015. p. 1-6.