IARITH: Uma Arquitetura Minimalista de Ponto Flutuante com ISA Reduzida para Métodos Numéricos

1st Marco Mello
Arquitetura de Computadores
Programa de Pós-Graduação em Ciência da Computação – UFABC
Santo André, Brasil
marcomello.e@gmail.com

***Abstract*—Este trabalho apresenta o projeto, implementação e avaliação da IARITH, uma arquitetura minimalista para processamento numérico em ponto flutuante. O projeto foi conduzido sob restrições severas: oito registradores de uso geral, 256 bytes de memória principal e um conjunto extremamente reduzido de instruções. A arquitetura foi implementada com 11 instruções, todas operando exclusivamente em ponto flutuante IEEE 754 (float32), sem suporte a inteiros. O objetivo é investigar o impacto de um ISA reduzido na implementação de algoritmos matemáticos complexos — como seno, cosseno, logaritmo e raiz n-ésima — que devem ser construídos inteiramente em software com base apenas nas operações aritméticas básicas fornecidas pelo hardware. A implementação do emulador em Python evidencia o papel fundamental da simulação na validação arquitetural e no co-design hardware–software. Os resultados demonstram a viabilidade da arquitetura e destacam os desafios inerentes ao design minimalista.**

***Index Terms*—Computer Architecture, ISA Design, Floating-Point, Numerical Methods, Emulator, RISC, Minimal Architecture.**

I. Introdução

O estudo de arquiteturas de computadores tradicionalmente se concentra nos princípios fundamentais que regem o design de processadores, sistemas de memória e conjuntos de instruções. A evolução histórica do campo é marcada por debates entre filosofias como RISC e CISC, que priorizam, respectivamente, simplicidade ou expressividade instrucional. Em ambientes educacionais, o desafio se intensifica quando os recursos disponíveis são deliberadamente reduzidos, exigindo um alinhamento rigoroso entre decisões de hardware e as necessidades do software.

Este trabalho se insere nesse contexto ao propor a **IARITH**, uma arquitetura minimalista voltada à computação numérica e construída inteiramente sobre operações em ponto flutuante de 32 bits. A IARITH é capaz de executar apenas um conjunto reduzido de operações básicas — soma, subtração, multiplicação e divisão — de modo que funções mais complexas, como seno, cosseno, logaritmo e raiz n-ésima, devem ser implementadas em software utilizando exclusivamente essas primitivas.

O projeto evidencia como decisões arquiteturais influenciam diretamente a complexidade do software e como a ausência deliberada de instruções especializadas desloca a responsabilidade computacional para o programador. Além disso, a construção de um emulador funcional demonstra a importância da simulação na validação arquitetural, prototipação e experimentação prática com ISAs customizadas.

II. Referencial Teórico

O desenvolvimento da arquitetura IARITH se fundamenta em princípios clássicos de projeto de processadores e em diretrizes consolidadas no estudo de ISAs minimalistas. A filosofia RISC, discutida por Patterson e Hennessy [1], destaca a importância de instruções simples, formatos uniformes e estruturas de execução previsíveis. Essa abordagem orientou diretamente a definição de um conjunto reduzido de instruções, com semântica clara e suficiente para a construção de métodos numéricos.

Outro pilar fundamental é o padrão IEEE 754 para operações em ponto flutuante. A IARITH utiliza exclusivamente imediatos de 32 bits codificados nesse formato, garantindo previsibilidade numérica e compatibilidade conceitual com arquiteturas reais. A documentação histórica do padrão, preservada em [2], fornece base para compreender representação, arredondamento e comportamento operacional dos valores float32 utilizados na ISA.

A combinação desses elementos — simplicidade arquitetural RISC e rigor matemático do IEEE 754 — constitui o alicerce conceitual da IARITH.

Além disso, arquiteturas minimalistas têm longa tradição em ambientes didáticos e experimentais, sendo frequentemente utilizadas para destacar a relação direta entre o conjunto de instruções e a complexidade do software. Exemplos clássicos incluem subconjuntos do MIPS, microarquiteturas simplificadas como LC-3 e processadores educacionais com poucas instruções aritméticas. A IARITH se insere nessa linha, mas explora um ponto de projeto ainda menos comum: uma ISA baseada exclusivamente em operações de ponto flutuante, sem suporte nativo a inteiros ou instruções lógicas. Essa escolha extrema evidencia como decisões arquiteturais moldam o espaço de soluções algorítmicas e reforçam o papel do co-design hardware–software na viabilidade de métodos numéricos em ambientes restritos.

III. Metodologia

A metodologia adotada neste trabalho foi estruturada em três eixos principais: (i) definição da arquitetura IARITH, (ii) implementação de um emulador funcional em Python, e (iii) desenvolvimento manual de rotinas numéricas em assembly. Cada etapa foi construída de maneira coerente com as restrições impostas pelo conjunto de instruções reduzido.

A. Projeto da Arquitetura

A arquitetura IARITH foi projetada seguindo princípios de simplicidade estrutural e uniformidade semântica. Os elementos fundamentais incluem oito registradores de uso geral contendo valores *float32*, 64 palavras de memória de 32 bits (256 bytes no total) e um conjunto de instruções baseado exclusivamente em operações de ponto flutuante.

Todas as instruções possuem um formato fixo de 45 bits contendo **sempre** um campo imediato de 32 bits:

$$\underbrace{\text{OPCODE}}_{4\text{ bits}} \mid \underbrace{\text{RD}}_{3\text{ bits}} \mid \underbrace{\text{RA}}_{3\text{ bits}} \mid \underbrace{\text{RB}}_{3\text{ bits}} \mid \underbrace{\text{IMMEDIATE}}_{32\text{ bits}}$$

A decisão de incluir o campo imediato em todas as instruções simplifica a decodificação e a geração de código, permitindo uma estrutura regular mesmo em uma ISA reduzida. A consequência prática é um comprimento não múltiplo de 8 bits, o que demandaria, em hardware físico, um mecanismo de alinhamento no estágio de *fetch* ou o empacotamento em um formato físico maior (por exemplo, 48 bits). O emulador abstrai essa complexidade.

A interpretação do campo imediato é dependente da instrução. O emulador produz simultaneamente:

- uma palavra de 32 bits (`imm_word`);
- um valor de ponto flutuante IEEE 754 (`imm_float`),

utilizando conversão binária em **big-endian**.

Na etapa de carregamento, imediatos associados a instruções aritméticas (`LDI`, `ADDi`) são tratados como *float32*, enquanto imediatos de instruções de memória e fluxo (`LDM`, `STM`, `JMP`, `JEQ`) são tratados como inteiros (com arredondamento quando necessário).

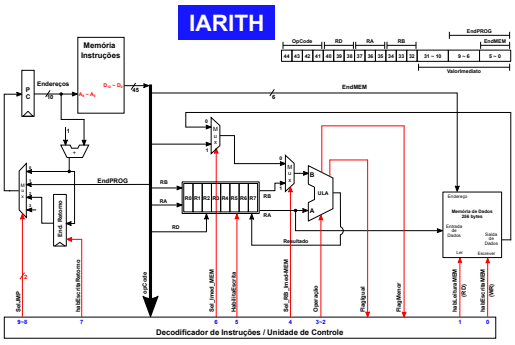

Fig. 1. Fluxo de dados da arquitetura IARITH, ilustrando o caminho de instruções, registradores, ULA, memória de dados e o formato unificado de instruções de 45 bits.

A Figura 1 apresenta o fluxo de dados completo da arquitetura, evidenciando a interação entre o contador de programa, memória de instruções, registradores, unidade aritmética e lógica (ULA) e memória de dados. O decodificador centraliza o controle de sinais e multiplexadores, enquanto o formato unificado de 45 bits permite tratar todas as instruções de forma homogênea. Esse diagrama serviu como base para o desenvolvimento do emulador e das rotinas numéricas apresentadas neste trabalho

A Tabela I resume as instruções disponíveis e seus campos.

Instr.	Semântica	Campos
LDI	RD ← IMM(float)	OP,RD,IMM
MOV	RD ← RB	OP,RD,RB
ADD	RD ← RA + RB	OP,RD,RA,RB
ADDi	RD ← RA + IMM	OP,RD,RA,IMM
SUB	RD ← RA - RB	OP,RD,RA,RB
MUL	RD ← RA × RB	OP,RD,RA,RB
DIV	RD ← RA / RB	OP,RD,RA,RB
STM	MEM[IMM] ← RA	OP,RA,IMM
LDM	RD ← MEM[IMM]	OP,RD,IMM
JMP	PC ← IMM	OP,IMM
JEQ	if RA==RB:PC←IMM	OP,RA,RB,IMM

TABLE I
Instruções da ISA IARITH.

Nota: nas instruções `LDM` e `STM`, alguns campos (como RB ou RD) são ignorados pelo decodificador devido ao formato unificado de 45 bits.

B. Implementação do Emulador

O emulador da IARITH foi desenvolvido em Python, com uma interface gráfica simples para facilitar a carga, execução e depuração de programas. Ele implementa o ciclo clássico de execução (busca, decodificação e execução) e reproduz fielmente a semântica das onze instruções da ISA, incluindo o tratamento diferenciado de imediatos e a comparação aproximada utilizada pela instrução `JEQ`.

O ambiente permite executar programas no formato assembly definido para a arquitetura, inspecionar registradores e memória em tempo real e acompanhar a evolução das rotinas numéricas descritas neste trabalho. O objetivo principal do emulador é oferecer uma plataforma prática para experimentação e validação arquitetural. O código completo encontra-se disponível no repositório oficial.

C. Implementação das Rotinas Numéricas

Para validar a expressividade do conjunto reduzido de instruções, quatro rotinas matemáticas fundamentais foram implementadas manualmente em assembly: $\sin(x)$ e $\cos(x)$, calculados por séries de Taylor; a raiz n-ésima por meio da iteração de Newton; e $\log_b(x)$ utilizando mudança de base com uma rotina auxiliar dedicada ao cálculo de $\ln(x)$.

Todas as rotinas foram escritas exclusivamente com as onze instruções da ISA IARITH, sem qualquer operação auxiliar ou suporte externo. Essa implementação evidencia como restrições arquiteturais — ausência de inteiros, comparação limitada a `JEQ`, falta de instruções lógicas ou de potência — moldam

diretamente as estratégias adotadas nos algoritmos, reforçando o caráter de co-design hardware–software da arquitetura.

O código completo das rotinas encontra-se disponível no repositório oficial.

D. Rotinas Numéricas: Métodos e Pseudocódigos

Esta subseção apresenta, para cada rotina numérica implementada, (i) o método matemático utilizado e (ii) o pseudocódigo correspondente, derivado diretamente das versões em assembly executadas no emulador.

1) Cálculo de $\sin(x)$*:* O cálculo do seno utiliza a expansão de Taylor em torno de zero:

$$\sin(x) = \sum_{k=0}^{\infty} (-1)^k \frac{x^{2k+1}}{(2k+1)!}.$$

A implementação em assembly adota uma forma iterativa do termo, evitando cálculo explícito de fatoriais e aproveitando a relação recursiva entre termos sucessivos.

```
# SENO(x) - Série de Taylor
term = x
sum  = x
for k in 0 .. N_it-1:
    denom  = (2*k+2)*(2*k+3)
    factor = -(x*x)/denom
    term   = term * factor
    sum    += term
# resultado = sum
```

2) Cálculo de $\cos(x)$*:* O cosseno segue a expansão:

$$\cos(x) = \sum_{k=0}^{\infty} (-1)^k \frac{x^{2k}}{(2k)!}.$$

Como no seno, os termos são calculados iterativamente, reutilizando x^2 para reduzir custo instrucional.

```
# COSSENO(x) - Série de Taylor
term = 1
sum  = 1
for k in 0 .. N_it-1:
    denom  = (2*k+1)*(2*k+2)
    factor = -(x*x)/denom
    term   = term * factor
    sum    += term
# resultado = sum
```

3) Raiz n*-ésima de* x*:* A raiz n-ésima é calculada usando a iteração de Newton:

$$y_{k+1} = \frac{1}{n}\left((n-1)y_k + \frac{x}{y_k^{n-1}}\right).$$

Devido ao ISA reduzido, y^{n-1} é obtido por multiplicações sucessivas, controladas por um contador em ponto flutuante.

```
# RAIZ n-ÉSIMA - Método de Newton
y = 1
for i in 1 .. N_it:
    p = 1
    for j in 1 .. n-1:
        p = p * y
    y = ((n-1)*y + x/p) / n
# resultado = y
```

4) Cálculo de $\log_b(x)$*:* O logaritmo é calculado por mudança de base:

$$\log_b(x) = \frac{\ln(x)}{\ln(b)}.$$

Assim, o núcleo da rotina é o cálculo de $\ln(u)$ para valores arbitrários. A implementação em assembly combina uma expansão interna com uma etapa de correção iterativa inspirada em Newton.

```
# LOG_b(x) - Mudança de base
ln_x = LN(x)
ln_b = LN(b)
return ln_x / ln_b

LN(u):
  g = chute_inicial
  for i in 1 .. N_it:
      S = 1
      p = 1
      for k in 1 .. M_terms:
          p = p * g
          S = S + p/k
      g = g + (u/S) - 1
  return g
```

5) Discussão Geral: Os pseudocódigos preservam elementos essenciais da implementação em assembly: contadores representados como *float32*, ausência de operações inteiras, cálculo iterativo de potências e armazenamento progressivo dos resultados parciais. As limitações da ISA — como a falta de instruções de comparação geral, de operações de potência ou de tipos inteiros — influenciam diretamente o estilo dos algoritmos, tornando-os valiosos como estudo de caso em co-design hardware–software.

O código-fonte completo das rotinas em assembly está disponível em:

https://github.com/Marco-Mello/IARITH

IV. Resultados

A execução das quatro rotinas no emulador confirmou a viabilidade da arquitetura IARITH para computação científica em ambiente restrito.

A. Precisão Numérica

As aproximações obtidas apresentaram erro absoluto típico inferior a 10^{-4} em comparação com valores de referência obtidos em `float64`. A semântica aproximada de `JEQ` mostrou-se adequada para controle de laços iterativos.

B. Custo Instrucional

O custo instrucional é elevado devido à natureza minimalista da ISA. Operações simples em arquiteturas convencionais requerem longas sequências de instruções, com predominância de multiplicações, divisões e saltos condicionais restritos.

C. Estabilidade e Convergência

Todas as rotinas apresentaram comportamento estável:

- **Seno e Cosseno**: convergência adequada para valores moderados.
- **Raiz n-ésima**: convergência monotônica em todas as iterações.
- **Logaritmo**: valores coerentes para $\ln(x)$ e $\log_b(x)$.

Reprodutibilidade

Todos os programas, rotinas em assembly e scripts utilizados neste trabalho estão disponíveis no repositório público:

https://github.com/Marco-Mello/IARITH

D. Síntese dos Resultados

Os experimentos demonstram:

- **Convergência numérica** compatível com ponto flutuante de precisão simples;
- **Elevado custo instrucional**, esperado para ISA reduzida;
- **Viabilidade** do uso da arquitetura para métodos numéricos;
- **Co-design hardware–software**: as soluções dependem diretamente das limitações e escolhas arquiteturais.

V. Conclusão

Este trabalho apresentou a IARITH, uma arquitetura minimalista orientada exclusivamente à computação numérica em ponto flutuante. A definição de uma ISA com apenas onze instruções, oito registradores e 256 bytes de memória demonstrou que, mesmo sob restrições extremas, é possível construir uma plataforma funcional para experimentação arquitetural e desenvolvimento de algoritmos matemáticos.

A implementação do emulador permitiu validar o comportamento da ISA e serviu como ferramenta essencial no processo de co-design hardware–software. As rotinas numéricas desenvolvidas evidenciaram que combinações de multiplicações, divisões e controle de fluxo restrito são suficientes para implementar funções matemáticas complexas, ainda que com custo instrucional elevado.

Como trabalhos futuros, destacam-se: (i) a inclusão de instruções adicionais de comparação ou operações compostas (como *fused multiply–add*), (ii) a exploração de diferentes modos de arredondamento IEEE 754 e (iii) a implementação de técnicas de redução de argumento para as rotinas trigonométricas. Em síntese, a IARITH demonstra que arquiteturas propositalmente reduzidas continuam sendo ferramentas valiosas para o estudo de fundamentos de ISA, experimentação de algoritmos e exploração de co-design hardware–software.

Por fim, este trabalho reforça que arquiteturas minimalistas continuam sendo ferramentas valiosas tanto em pesquisa quanto em ensino. A IARITH evidencia, de forma clara, como a redução extrema do conjunto de instruções desloca responsabilidades para o software e revela o custo real de algoritmos numéricos quando privados de instruções matemáticas complexas. Esse caráter exploratório e transparente torna a arquitetura especialmente adequada para disciplinas de arquitetura de computadores, organização de processadores e fundamentos de ponto flutuante, além de abrir espaço para investigações mais profundas sobre expressividade, análise de custo instrucional e limites do design de ISAs. Assim, a IARITH não apenas cumpre seu papel como plataforma experimental, mas também estabelece uma base sólida para futuras extensões e estudos sobre minimalidade arquitetural.

VI. Referências

[1] D. Patterson and J. Hennessy, *Computer Organization and Design RISC-V Edition: The Hardware/Software Interface*. Morgan Kaufmann, 1st Edition, 2017.

[2] IEEE 754 Floating-Point Standard — Historical References. Disponível em: https://web.archive.org/web/20111201211023/http://babbage.cs.qc.cuny.edu/IEEE-754.old/References.xhtml