

## Arquitetura de Computadores / Programação Assembly

Custo



Simplicidade



Desempenho



Elegância



Espaço para crescimento



Tamanho de Programa



Isolamento de Arquitetura



Facilidade de programação

O Guia Prático RISC-V é uma concisa introdução e referência para programadores de sistemas embarcados, estudantes e aos curiosos sobre uma arquitetura moderna, popular e aberta. O RISC-V abrange desde o microcontrolador de 32 bits de baixo custo até o mais rápido computador na nuvem de 64 bits.

Ao utilizar os recursos visuais ilustrados acima, o texto mostra como o RISC-V incorporou as boas ideias de arquiteturas passadas, evitando os erros cometidos.

- Introduz o RISC-V em apenas 100 páginas, incluindo 75 figuras
- Cartão de Referência RISC-V de 2 páginas que resume todas as instruções
- Glossário de Instruções de 50 páginas que define todas as instruções detalhadamente
- 75 dicas de boas práticas de projeto de arquitetura utilizando os ícones acima
- 50 barras laterais com comentários interessantes e com o histórico do RISC-V
- 25 citações para transmitir o conhecimento de cientistas e engenheiros notáveis

Dez capítulos apresentam cada componente do conjunto de instruções modular RISC-V, muitas vezes, contrastando código compilado de C para RISC-V versus as arquiteturas ARM, Intel e MIPS, porém os leitores podem iniciar a programação após o Capítulo 2.

*"I like RISC-V and this book as they are elegant—brief, to the point, and complete."* — C. Gordon Bell



David Patterson (Google & UC Berkeley) e

Andrew Waterman (SiFive) são 2 dos 4 arquitetos RISC-V



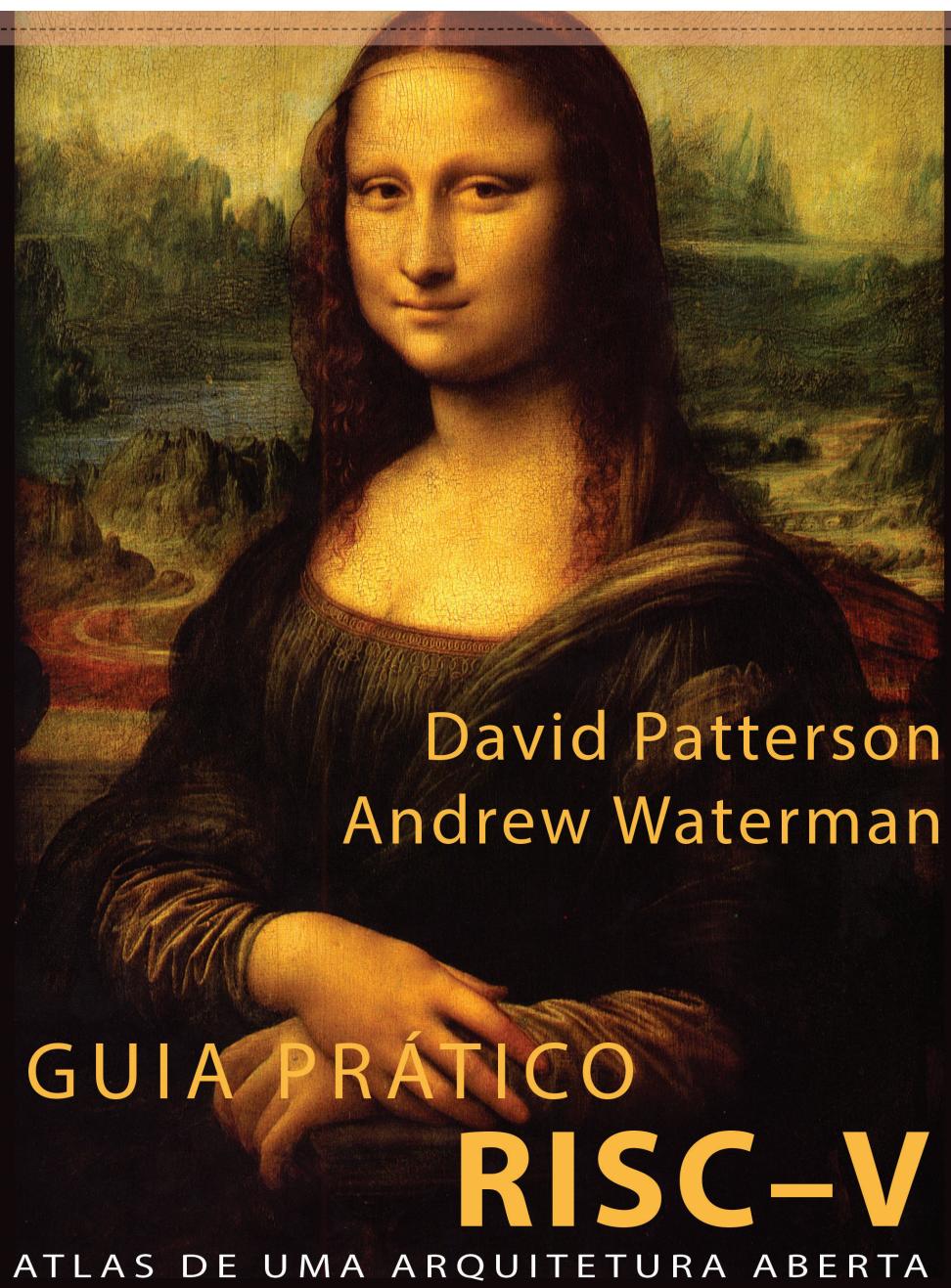
Strawberry Canyon LLC  
San Francisco, CA, USA



Guia Prático RISC-V

1<sup>st</sup>

Patterson & Waterman



# 1

# Por que RISC-V?

**Leonardo da Vinci**  
(1452-1519) foi um arquiteto, engenheiro, escultor renascentista e pintor da Mona Lisa.



*Simplicity is the ultimate sophistication.*

—Leonardo da Vinci

## 1.1 Introdução

O objetivo do RISC-V (“RISC five”) é tornar-se uma arquitetura de conjunto de instruções universal (*Instruction Set Architecture*—ISA). Para isso, ele deve satisfazer alguns requisitos:

- Atender a todos os tamanhos de processadores, desde o minúsculo controlador embarcado até o computador de alto desempenho mais rápido.
- Funcionar bem com uma grande variedade de softwares e linguagens de programação populares.
- Acomodar todas as tecnologias de implementação: FPGAs (Field-Programmable Gate Arrays), ASICs (Application-Specific Integrated Circuits), chips customizados e até mesmo futuras tecnologias de dispositivos.
- Ser eficiente para todos os tipos de microarquitetura: controle microcodificado ou hard-wired; pipelines em ordem, desacoplados ou desordenados; emissão de instrução única ou superescalar; e assim por diante.
- Apoiar ampla especialização para atuar como uma base para aceleradores customizados, que aumentam em importância à medida que a Lei de Moore se desvanece.
- Ser estável, ou seja, a ISA base não deve mudar. Mais importante, a ISA não pode ser descontinuada, como aconteceu no passado com as ISAs proprietárias, como a AMD Am29000, a Digital Alpha, a Digital VAX, o Hewlett Packard PA-RISC, Intel i860, Intel i960, Motorola 88000, e a Zilog Z8000.

O RISC-V é incomum não apenas porque é uma ISA recente—nascida nesta década quando a maioria das alternativas data dos anos 1970 ou 1980—mas também porque é uma ISA *aberta*. Ao contrário de praticamente todas as arquiteturas anteriores, seu futuro é livre do destino ou dos caprichos de qualquer empresa, que condenou muitas ISAs no passado.

>\$50B		>\$5B, <\$50B		>\$0.5B, <\$5B	
Google	USA	BAE Systems	UK	AMD	USA
Huawei	China	MediaTek	Taiwan	Andes Technology	China
IBM	USA	Micron Tech.	USA	C-SKY Microsystems	China
Microsoft	USA	Nvidia	USA	Integrated Device Tech.	USA
Samsung	Korea	NXP Semi.	Netherlands	Mellanox Technology	Israel
		Qualcomm	USA	Microsemi Corp.	USA
		Western Digital	USA		

**Figura 1.1:** Os membros corporativos da Fundação RISC-V de acordo com o sexto Workshop RISC-V em maio de 2017, classificado por vendas anuais. As empresas da coluna da esquerda excedem as vendas anuais de \$US 50B, as empresas da coluna intermediária vendem menos de \$US 50B, mas mais de \$US 5B, e as vendas da coluna da direita são menores que \$US 5B, mas de ordem maior que \$US 0.5B. A fundação inclui outras 25 empresas menores, 5 empresas iniciantes (Antmicro Ltd, Blockstream, Esperanto Technologies, Greenwaves Technologies e SiFive), 4 organizações sem fins lucrativos (CSEM, Draper Laboratory, ICT e lowRISC) e 6 universidades (ETH Zurich, IIT Madras, National University of Defense Technology, Princeton e UC Berkeley). A maioria das 60 organizações tem sua sede fora dos EUA. Para saber mais, acesse [www.riscv.org](http://www.riscv.org).

Em vez disso, pertence a uma fundação aberta e sem fins lucrativos. O objetivo da RISC-V Foundation é manter a estabilidade do RISC-V, evoluí-lo lenta e cuidadosamente, apenas por razões técnicas, e tentar torná-lo tão popular para o hardware quanto o Linux é para sistemas operacionais. Como sinal de sua vitalidade, a Figura 1.1 lista os maiores membros corporativos da Fundação RISC-V.

## 1.2 ISAs Modulares vs. Incrementais

*A Intel estava apostando seu futuro em um microprocessador de alta qualidade, mas ainda faltavam anos para isso. Para combater a Zilog, a Intel desenvolveu um processador stop-gap e o chamou de 8086. O objetivo era ter vida curta e não ter sucessores, mas não foi assim que as coisas aconteceram. O processador de ponta acabou chegando ao mercado e, quando saiu, ficou muito lento. Então, a arquitetura do 8086 continuou viva - evoluiu para um processador de 32 bits e, eventualmente, para um processador de 64 bits. Os nomes continuavam mudando (80186, 80286, i386, i486, Pentium), mas o conjunto de instruções permaneceu intacto.*

—Stephen P. Morse, arquiteto do 8086 [Morse 2017]

A abordagem convencional para arquitetura de computadores é o desenvolvimento de ISAs *incrementais*, onde novos processadores devem implementar não apenas novas extensões ISA, mas também todas as extensões do passado. O objetivo é manter a *compatibilidade binária retroativa* para que versões binárias de programas de décadas possam ser executadas corretamente no processador mais recente. Esse requisito, quando combinado com o apelo comercial de anunciar novas instruções em uma nova geração de processadores, levou a ISAs que crescem substancialmente em tamanho com a idade. Por exemplo, a Figura 1.2 mostra o crescimento no número de instruções para uma ISA dominante hoje: o 80x86. Ela remonta a 1978, mas adicionou aproximadamente *três instruções por mês* ao longo de sua vida útil.

Essa convenção significa que toda implementação do x86-32 (o nome que usamos para a versão de endereço de 32 bits do x86) deve implementar os erros das extensões anteriores, mesmo quando elas não fazem mais sentido. Por exemplo, a Figura 1.3 descreve a instrução *ASCII Adjust after Addition* (aaa) do x86, que há muito sobrevive com sua inutilidade.

**Adicionamos barras laterais às margens** para oferecer comentários relevantes. Por exemplo, o RISC-V foi originalmente desenvolvido para uso interno em pesquisas e cursos da UC Berkeley. Tornou-se aberto porque as pessoas de fora começaram a usá-lo por conta própria. Os arquitetos do RISC-V aprenderam sobre o interesse externo quando começaram a receber reclamações sobre as mudanças da ISA em seu curso, que estava na web. Somente depois que os arquitetos entenderam a necessidade, tentaram torná-lo um padrão aberto de ISA.

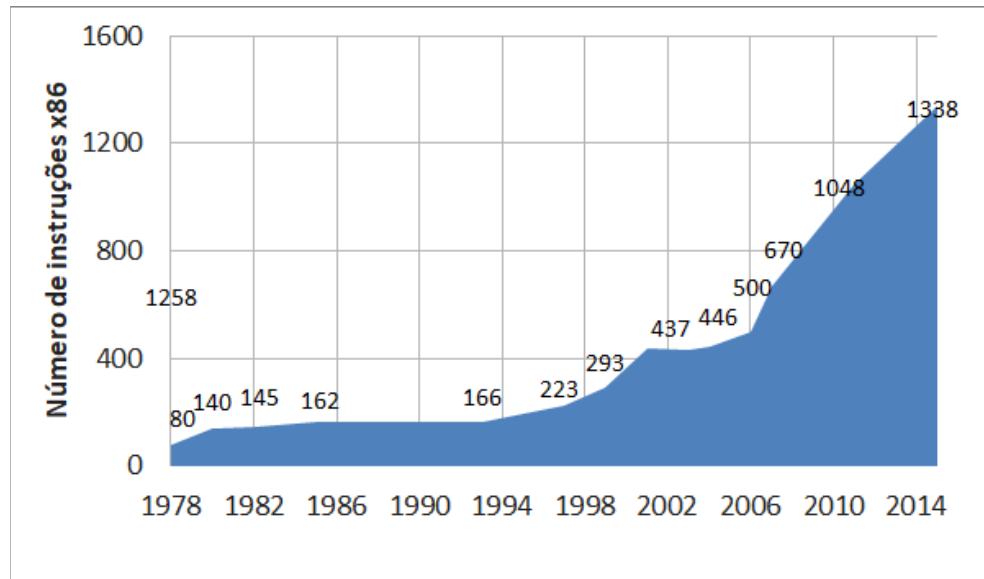


Figura 1.2: Crescimento do conjunto de instruções x86 ao longo de sua vida. O x86 iniciou com 80 instruções em 1978. Aumentou as instruções em uma fração de 16X para 1338 em 2015 e continua crescendo. Surpreendentemente, este gráfico é conservador. Um blog da Intel estima a contagem em 3600 instruções em 2015 [Rodgers and Uhlig 2017], o que aumentaria a taxa do x86 para uma nova instrução *a cada quatro dias* entre 1978 e 2015. Em nosso cálculo contabilizamos somente as instruções de linguagem assembly, e eles possivelmente acrescentaram a contagem de instruções de linguagem de máquina. Como o Capítulo 8 explica, uma grande parte do crescimento é porque a ISA x86 baseia-se em instruções SIMD para o paralelismo em nível de dados.

```

O registrador AL é a origem e o destino padrão.
Se os 4 bits mais baixos do registrador AL forem > 9,
    ou o transportador de flag auxiliar AF = 1,
Então
    Adicione 6 aos 4 bits mais baixos de AL e descarte o overflow
    Incrementa o byte mais alto de AL
    Carry flag CF = 1
        transportador de flag auxiliar AF = 1
Senão
    CF = AF = 0
4 bits mais altos de AL = 0

```

Figura 1.3: Descrição da instrução x86-32 ASCII Adjust after Addition (aaa). Esta executa a aritmética computacional em Binary Coded Decimal (BCD), representação numérica que já caiu na lixeira da história da tecnologia. O x86 também possui três instruções relacionadas para subtração (aas), multiplicação (aam) e divisão (aad). Como cada uma é uma instrução de um byte, elas ocupam coletivamente 1.6 %(4/256) do precioso espaço de opcode.

Como uma analogia, suponha que um restaurante sirva apenas uma refeição a preço fixo, que inicia por um pequeno jantar de hambúrguer e milkshake. Com o tempo, acrescenta-se batatas fritas e depois um *sundae* de sorvete, seguido de salada, torta, vinho, massa vegetariana, bife, cerveja, *ad infinitum* até se tornar um grande banquete. Pode fazer pouco sentido no total, mas os clientes podem encontrar o que já comeram em uma refeição passada naquele restaurante. A má notícia é que os clientes devem pagar o aumento do custo da expansão do banquete para cada jantar.

Além de ser recente e aberto, o RISC-V é incomum, pois, ao contrário de quase todas as ISAs anteriores, é *modular*. No núcleo há um ISA básico, chamado *RV32I*, que executa uma pilha completa de software. O *RV32I* está congelado e nunca será alterado, o que dá aos criadores de compiladores, desenvolvedores de sistemas operacionais e programadores de linguagem assembly um destinoável. A modularidade vem de extensões padrão opcionais que o hardware pode incluir ou não, dependendo das necessidades da aplicação. Essa modularidade permite implementações muito pequenas e de baixo consumo de energia, que podem ser críticas para aplicativos embarcados. Informando ao compilador RISC-V quais extensões estão incluídas, ele pode gerar o melhor código para esse hardware. A convenção é anexar as letras de extensão ao nome para indicar quais estão incluídas. Por exemplo, o *RV32IMFD* adiciona as extensões de multiplicação de ponto flutuante (*RV32M*), de precisão simples (*RV32F*) e de ponto flutuante de precisão dupla (*RV32D*) às instruções de base obrigatórias (*RV32I*).

Voltando à nossa analogia, o RISC-V oferece um menu em vez de um buffet; o chef precisa cozinhar apenas o que os clientes querem—não um banquete para cada refeição—e os clientes pagam apenas pelo que pedem. O RISC-V não precisa adicionar instruções simplesmente para o marketing sizzle. A RISC-V Foundation decide quando deve-se adicionar uma nova opção ao menu, e eles o farão apenas por razões técnicas sólidas após uma ampla discussão aberta por um comitê de especialistas em hardware e software. Mesmo quando novas opções aparecem no menu, elas permanecem opcionais e não são um novo requisito para todas as implementações futuras, como acontece com ISAs incrementais.

### 1.3 ISA Design 101

Antes de introduzir a ISA RISC-V, será útil entender os princípios e as escolhas que um arquiteto de computador deve fazer ao projetar uma ISA. Abaixo está uma lista das sete medidas, junto com os ícones que colocaremos nas margens da página para destacar os momentos em que RISC-V as aborda nos capítulos seguintes. (A contracapa do livro impresso tem uma legenda para os ícones.)

- custo (ícone da moeda de dólar)
- simplicidade (roda)
- desempenho (velocímetro)
- isolamento de arquitetura da implementação (metades separadas de um círculo)
- espaço para crescimento (acordeão)
- tamanho do programa (setas opostas)
- facilidade de programação / compilação / “linkagem” (“tão fácil quanto o ABC”).

**Se o software usar uma instrução RISC-V omitida de uma extensão opcional, é gerado um alerta e executada a função desejada no software como parte de uma biblioteca padrão.**



Custo



Simplicidade



Desempenho



Isolamento de Arq. da Impl.



Espaço para Crescimento



Tamanho do Programa



Facilidade de Programação

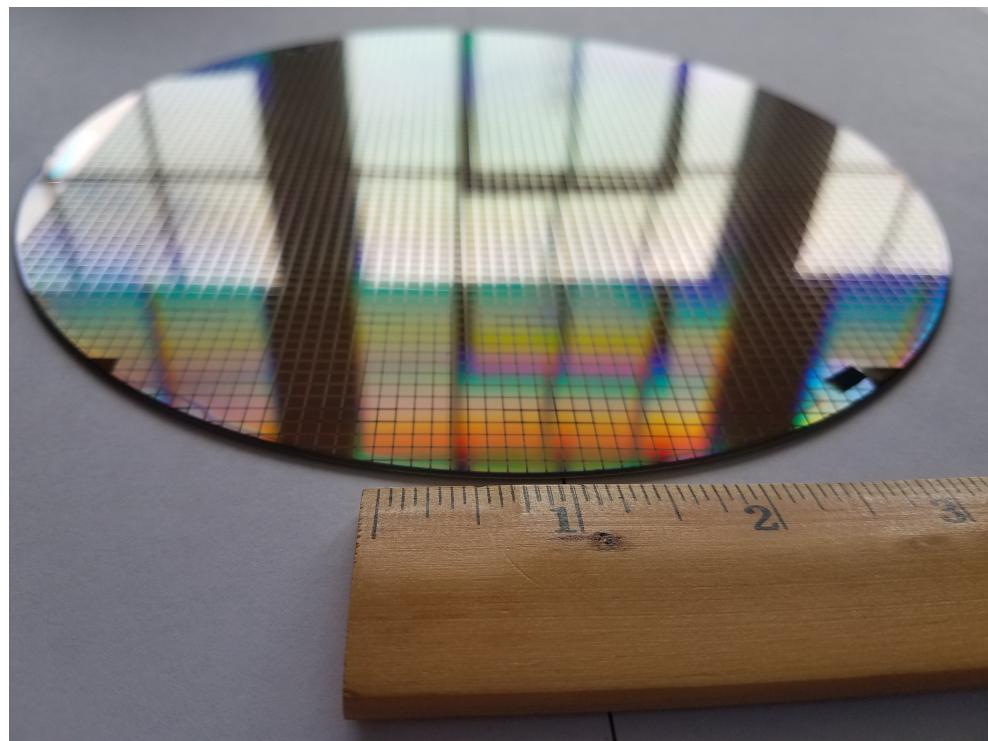


Figura 1.4: Um wafer de 8 polegadas de diâmetro de *dies*(recortes) RISC-V projetado pela SiFive. Possui dois tipos de *dies* RISC-V usando uma linha de processamento maior e mais antiga. Um molde FE310 é  $2.65\text{ mm} \times 2.72\text{ mm}$  e um *die* de teste SiFive que é  $2.89\text{ mm} \times 2.72\text{ mm}$ . O wafer contém 1846 do primeiro e 1866 do último, totalizando 3712 chips.

Para ilustrar o que queremos dizer, nesta seção mostraremos algumas escolhas feitas em ISAs mais antigas que parecem imprudentes ao analisarmos em retrospectiva, e em contrapartida, onde o RISC-V tomou decisões muito melhores.

**Custo.** Os processadores são implementados como circuitos integrados, comumente chamados de *chips* ou *dies*(recortes). Eles são chamados de *dies* porque iniciam como um pedaço de um único wafer redondo, que é *recortado* em muitas peças individuais. A Figura 1.4 mostra um wafer de processadores RISC-V. O custo é muito sensível à área dos *dies*:

$$\text{custo} \approx f(\text{die area}^2)$$

Obviamente, quanto menor o *die*, mais *dies* são fabricados por wafer e a maior parte do custo do chip é a própria wafer processada. Menos óbvio é que quanto menor o *die*, maior o rendimento da produção, a fração de chips fabricados que funcionam. A razão é que a fabricação de silício resulta em pequenas falhas espalhadas sobre o wafer, então quanto menor a área, menor a fração que será defeituosa.

Um arquiteto deseja manter a ISA simples para reduzir o tamanho dos processadores que o implementam. Como veremos nos próximos capítulos, a ISA RISC-V é muito mais simples que o ISA ARM-32. Como um exemplo concreto do impacto da simplicidade, compararamos um processador RISC-V Rocket a um processador ARM-32 Cortex-A5 na mesma tecnologia (TSMC40GPLUS) usando as caches de mesmo tamanho (16 KiB). O *die* RISC-V é 0.27 mm<sup>2</sup> versus 0.53 mm<sup>2</sup> para o ARM-32. Em torno de duas vezes a área, o ARM-32 Cortex-A5 custa aproximadamente 4X (2<sup>2</sup>) mais que o RISC-V Rocket. Mesmo um *die* menor em 10% reduz o custo por um fator de 1,2 (1,1<sup>2</sup>).

**Processadores high-end** podem ganhar desempenho combinando instruções simples, sem sobrecarregar todas as implementações de baixo custo com uma ISA maior e mais complicada. Essa técnica é chamada *macro-fusion*, pois ela combina “macro” instruções.

**Simplicidade.** Dada a sensibilidade do custo à complexidade, os arquitetos desejam uma ISA simples para reduzir a área de impressão. A simplicidade também reduz os tempos de projeto e verificação de chips, o que pode representar muito do custo de desenvolvimento do chip. Esses custos devem ser adicionados ao custo final do produto, dependendo do número de chips vendidos. A simplicidade também reduz o custo da documentação e a dificuldade de fazer com que os clientes entendam como usar o ISA.

Abaixo encontra-se um exemplo da complexidade do ISA ARM-32:

```
ldmiaeq SP!, {R4-R7, PC}
```

A instrução significa *Load Multiple, Increment-Address, on EQual*. Ela executa 5 cargas de dados e escreve em 6 registradores, mas executa somente se a condição EQ for satisfeita. Além disso, ela escreve um dos resultados no PC, por isso a instrução também executa um desvio condicional. Quanta utilidade!

**Um processador simples pode ser útil para aplicativos embarcados** já que é mais fácil prever o tempo de execução. Os programadores de linguagem assembly de micro-controladores geralmente querem manter o controle de tempo exato, então eles confiam no código que leva um número previsível de ciclos de clock que eles podem contar manualmente.

Ironicamente, as instruções simples são muito mais prováveis de serem usadas do que as complexas. Por exemplo, o x86-32 inclui uma instrução *enter*, que deveria ser a primeira instrução executada ao entrar em um procedimento para criar um quadro de pilha (*stack frame*)(veja o Capítulo 3). A maioria dos compiladores usa apenas estas duas instruções simples do x86-32:

```
push ebp      # Coloca o ponteiro de quadros na pilha
mov ebp, esp # Copia o ponteiro da pilha para o ponteiro do quadro
```

**Desempenho.** Com exceção dos pequenos chips para aplicações embarcadas, os arquitetos normalmente se preocupam com desempenho e custo. O desempenho pode ser consider-



Simplicidade



Desempenho

ado em três termos:

$$\frac{\text{instrucoes}}{\text{programa}} \times \frac{\text{media ciclos clock}}{\text{instrucao}} \times \frac{\text{tempo}}{\text{ciclo clock}} = \frac{\text{tempo}}{\text{programa}}$$

**O último termo é o inverso da taxa de clock**, então uma taxa de clock de 1 GHz significa que o tempo por ciclo de clock é de 1 ns ( $1/10^9$ ).

**O número médio de ciclos de clock pode ser menor que 1** porque o A9 e o BOOM [Celio et al. 2015] são chamados de processadores *superscalar*, que executam mais de uma instrução por ciclo de clock.

Mesmo que uma ISA simples possa executar mais instruções por programa do que uma ISA complexa, ela pode compensar isso com um ciclo de clock mais rápido ou uma média menor de ciclos de clock por instrução (CPI).

Por exemplo, para o benchmark CoreMark [Gal-On and Levy 2012] (100.000 iterações), o desempenho no ARM-32 Cortex-A9 é

$$\frac{32.27 B \text{ instrucoes}}{\text{programa}} \times \frac{0.79 \text{ ciclos clock}}{\text{instrucao}} \times \frac{0.71 \text{ ns}}{\text{ciclos clock}} = \frac{18.15 \text{ seg}}{\text{programa}}$$

Para a implementação BOOM do RISC-V, a equação é

$$\frac{29.51 B \text{ instrucoes}}{\text{programa}} \times \frac{0.72 \text{ ciclos clock}}{\text{instrucao}} \times \frac{0.67 \text{ ns}}{\text{ciclos clock}} = \frac{14.26 \text{ seg}}{\text{programa}}$$

O processador ARM não executou menos instruções do que o RISC-V nesse caso. Como veremos, as instruções simples também são as instruções mais populares, de modo que a simplicidade da ISA pode superar em todas as métricas. Para este programa, o processador RISC-V ganha quase 10% em cada um dos três fatores, o que resulta em uma vantagem de desempenho de quase 30%. Se uma ISA mais simples também resultar em um chip menor, sua fração custo/benefício será excelente.

**Isolamento de Arquitetura da Implementação.** A distinção original entre *arquitetura* e *implementação*, que remete à década de 1960, é que arquitetura é o que um programador de linguagem assembly precisa saber para escrever um programa correto, mas sem se preocupar com o desempenho desse programa. A tentação de um arquiteto é incluir instruções em um ISA que auxiliem o desempenho ou o custo de uma implementação em um determinado momento, mas sobrecarregue implementações diferentes ou futuras.

Para a ISA MIPS-32, o exemplo lamentável foi o *delayed branch*. Desvios condicionais causam problemas na execução em pipeline porque o processador deseja ter a próxima instrução a ser executada no pipeline, mas não consegue decidir se deseja a próxima instrução sequencial (se o desvio não for tomado) ou a que está no endereço de destino do desvio (se for tomado). Para seu primeiro microprocessador com um pipeline de 5 estágios, essa indecisão poderia ter causado uma parada no ciclo do pipeline. O MIPS-32 resolveu esse problema redefinindo o desvio para ocorrer na instrução *após* a próxima. Assim, a instrução seguinte é *sempre* executada. A tarefa do programador ou do compilador era colocar algo útil no intervalo de atraso (*delay slot*).

Infelizmente, essa “solução” não ajudou mais tarde os processadores MIPS-32 com muito mais estágios de pipeline (com muitas outras instruções buscadas antes do resultado do desvio ser computado), e ainda tornou a vida mais difícil para os programadores do MIPS-32, desenvolvedores de compiladores, e projetistas de processadores, pois ISAs incrementais exigem compatibilidade retroativa (veja Seção 1.2). Além disso, torna o código MIPS-32 muito mais difícil de entender (veja a Figura 2.10 na Página 32).

Enquanto arquitetos não devem colocar recursos que *ajudam* apenas uma implementação em um determinado momento, esses também não devem colocar features que *complicam* algumas implementações. Por exemplo, o ARM-32 e alguns outros ISAs têm uma instrução



Isolamento de Arq. da Impl.

**Processadores com pipelines hoje antecipam os resultados dos desvios** usando preditores de hardware, que podem exceder a precisão de 90% e trabalhar com qualquer tamanho de pipeline. Eles precisam apenas de um mecanismo para liberar e reiniciar o pipeline quando falham na predição.

Load Multiple, conforme mencionado na página anterior. Essas instruções podem melhorar o desempenho de projetos com pipeline de emissão de instrução simples, mas prejudicam os pipelines de instruções múltiplas. O motivo é que a implementação direta impede o planejamento das cargas individuais de um Load Multiple em paralelo com outras instruções, reduzindo o “throughput” de instrução de tais processadores.

**Espaço para Crescimento.** Com o fim da Lei de Moore, o único caminho a seguir para grandes melhorias no custo-desempenho é adicionar instruções personalizadas para domínios específicos, como *deep learning*, realidade aumentada, otimização combinatória, gráficos e assim por diante. Isso significa que é importante hoje que uma ISA reserve espaço de opcode para futuras melhorias.

Nas décadas de 1970 e 1980, quando a Lei de Moore estava em pleno vigor, não se pensava em economizar espaço de opcode para aceleradores futuros. Em vez disso, os arquitetos valorizavam o endereço maior e os campos imediatos para reduzir o número de instruções executadas por programa, o primeiro termo na equação de desempenho na página anterior.

Um exemplo do impacto da escassez de espaço de opcode foi quando os arquitetos do ARM-32 mais tarde tentaram reduzir o tamanho do código adicionando instruções de 16 bits à ISA de 32 bits anteriormente uniforme. Simplesmente não havia mais espaço. Assim, a única solução foi criar uma nova ISA primeiro com instruções de 16 bits (Thumb) e depois uma nova ISA com instruções de 16 bits e 32 bits (Thumb-2) usando um bit de modo para alternar entre ISAs ARM. Para alterar os modos, o programador ou compilador desvia para um endereço de byte com um 1 no bit menos significativo, o que funcionou porque as instruções de 16 e 32 bits possuem 0 nesse bit.

**Tamanho do Programa.** Quanto menor o programa, menor a área necessária em um chip para a memória do programa, o que pode representar um custo significativo para dispositivos embarcados. Na verdade, esse problema inspirou arquitetos ARM a adicionar retroativamente instruções mais curtas nas ISAs Thumb e Thumb-2. Programas menores também levam a menos *cache miss* de instruções, o que economiza energia, já que os acessos DRAM fora do chip usam muito mais energia do que os acessos SRAM no chip, e por isso também apresentam um melhor desempenho. Tamanho pequeno de código é um dos principais objetivos dos arquitetos de ISAs.

A ISA x86-32 tem instruções tão curtas quanto 1 e até 15 bytes. Seria de se esperar que as instruções de comprimento de variável em byte do x86 certamente levassem a programas menores do que as ISAs limitadas a instruções de comprimento de 32 bits, como o ARM-32 e o RISC-V. Logicamente, as instruções de tamanho variável de 8 bits também devem ser menores que as ISAs que oferecem apenas instruções de 16 bits e 32 bits, como Thumb-2 e RISC-V usando a extensão RV32C (consulte o Capítulo 7). A Figura 1.5 mostra que, enquanto o código ARM-32 e RISC-V é 6% a 9% maior que o código para x86-32 quando todas as instruções são 32 bits, surpreendentemente x86-32 é 26% maior que as versões compactadas (RV32C e Thumb-2) que oferecem instruções de 16 bits e 32 bits.

Enquanto uma nova ISA usando instruções variáveis de 8 bits provavelmente levaria a um código menor que o RV32C e o Thumb-2, os arquitetos da primeira versão do x86 nos anos 70 tinham preocupações diferentes. Além disso, dado o requisito de retrocompatibilidade de um ISA incremental (Seção 1.2), as centenas de novas instruções x86-32 são mais longas do que se poderia esperar, uma vez que elas carregam o ônus de um prefixo de um ou dois bytes para comprimi-los no espaço livre, de opcode restrito e disponível do x86 original.

**Facilidade de Programação, Compilação, e “Linkagem”.** Como os dados em um registrador são muito mais rápidos de serem acessados do que os dados na memória, é im-



Espaço para Crescimento

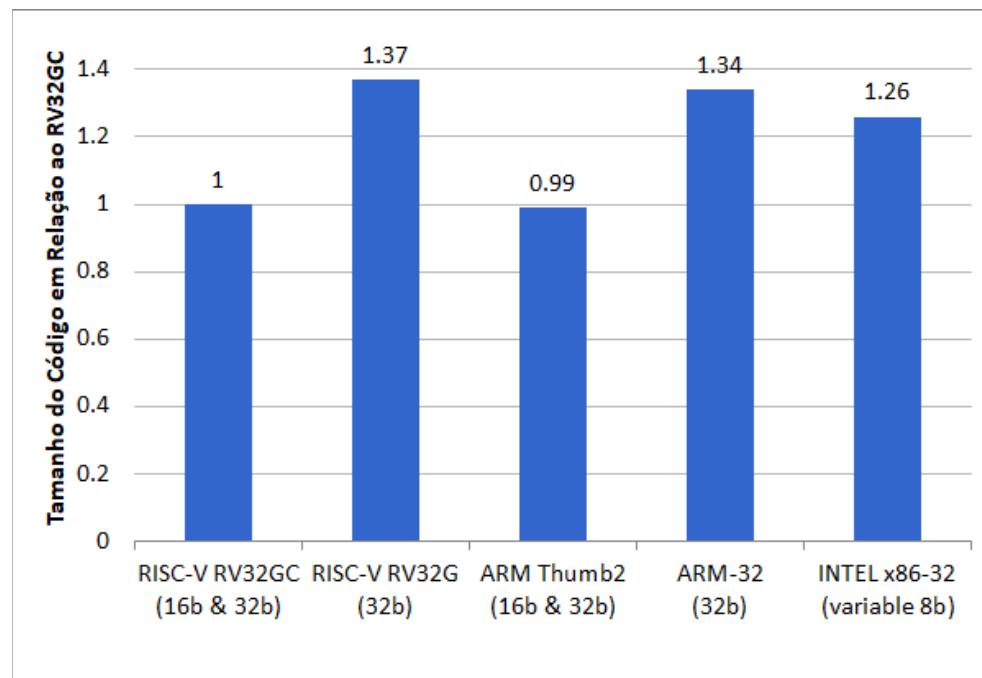
**A instrução ARM-32 1dmiaeq mencionada acima é ainda mais complicada**, porque quando realiza um desvio também pode alterar os modos de conjunto de instruções entre o ARM-32 e Thumb/Thumb-2.



Tamanho do Programa

Um exemplo de instrução x86-32 de 15 bytes é lock add dword ptr ds:[esi+ecx\*4 +0x12345678], Oxefcdab89. Monta em (em hexadecimal): 67 66 f0 3e 81 84 8e 78 56 34 12 89 ab cd ef. Os últimos 8 bytes são 2 endereços e os primeiros 7 bytes representam a *atomic memory operation*, a operação add, dados de 32 bits, o registrador de segmento de dados, os dois registradores de endereço e o modo de endereçamento indexado. Um exemplo de instrução de 1 byte é inc eax que é convertido em em 40.

**OPC**  
Facilidade de Programação



**Figura 1.5:** Tamanhos relativos de programa para RV32G, ARM-32, x86-32, RV32C e Thumb-2. As duas últimas ISAs destinam-se ao tamanho de código pequeno. Os programas são os benchmarks SPEC CPU2006 usando compiladores GCC. A pequena vantagem de tamanho do Thumb-2 sobre o RV32C deve-se à economia em tamanho de código de Load and Store Multiple na entrada do procedimento. O RV32C os exclui para manter o mapeamento um-para-um para as instruções do RV32G, que omite Load and Store Multiple para reduzir a complexidade da implementação em processadores high-end (veja abaixo). O Capítulo 7 descreve o RV32C. O RV32G indica uma combinação popular de extensões RISC-V (RV32M, RV32F, RV32D e RV32A), adequadamente denominada RV32IMAFD. [Waterman 2016]

portante que os compiladores realizem um bom trabalho na alocação de registradores. Essa tarefa é muito mais fácil quando há muitos registradores em vez de poucos. Sob essa ótica, o ARM-32 tem 16 registradores e o x86-32 tem apenas 8. A maioria das ISAs disponíveis, incluindo o RISC-V, tem um número relativamente generoso de 32 registradores inteiros. Mais registradores certamente facilitam a vida de compiladores e programadores de linguagem assembly.

Outro problema para compiladores e programadores assembly é descobrir a velocidade de uma sequência de código. Como veremos, as instruções RISC-V normalmente são de no máximo um ciclo de clock por instrução (ignorando *cache miss*), enquanto como vimos anteriormente, tanto o ARM-32 quanto o x86-32 possuem instruções que levam muitos ciclos de clock mesmo quando tudo se encaixa na cache. Além disso, ao contrário do ARM-32 e do RISC-V, as instruções aritméticas x86-32 podem ter operandos na memória, em vez de exigir que todos os operandos estejam nos registradores. Instruções complexas e operandos na memória dificultam que os projetistas de processadores forneçam previsibilidade de desempenho.

É útil para uma ISA ter suporte a código independente de posição (*position independent code — PIC*), porque este permite “linkagem” dinâmica (veja a Seção 3.5), já que o código da biblioteca compartilhada pode residir em endereços diferentes em programas diferentes. Os desvios relativos ao PC e o endereçamento de dados são uma vantagem para o PIC. Enquanto quase todas as ISAs fornecem desvios relativos ao PC, x86-32 e MIPS-32 omitem todo endereçamento de dados relativo ao PC.

---

**■ Elaboração: ARM-32, MIPS-32, e x86-32**

---

Elaborações são seções opcionais nas quais os leitores podem aprofundar-se mais caso estiverem interessados em um tópico, mas você não precisa lê-las para entender o restante do livro. Por exemplo, nossos nomes de ISA não são os oficiais. A ISA ARM de 32 bits tem muitas versões, sendo a primeira em 1986 e a mais recente chamada ARMv7 em 2005. ARM-32 geralmente se refere à ISA ARMv7. O MIPS também tinha muitas versões de 32 bits, mas estamos nos referindo ao original, chamado MIPS I (“MIPS32” é uma ISA posterior e diferente do que chamamos de MIPS-32). A primeira arquitetura de endereços de 16 bits da Intel foi o 8086 em 1978, que a ISA 80386 expandiu para endereços de 32 bits em 1985. Nossa notação x86-32 geralmente se refere ao IA-32, a versão com endereço de 32 bits da ISA x86. Dada a extensa quantidade de variantes dessas ISAs, acreditamos que nossa terminologia fora do padrão menos confusa.

---

## 1.4 Uma Visão Geral deste Livro

Este livro pressupõe que você já tenha visto outros conjuntos de instruções antes do RISC-V. Caso não tenha, considere dar uma olhada em nosso livro introdutório de arquitetura introduitoria com base no RISC-V [Patterson and Hennessy 2017].

O Capítulo 2 introduz o RV32I, a base “congelada” de instruções com inteiros que são o coração do RISC-V. O Capítulo 3 descreve a linguagem assembly RISC-V restante além daquela introduzida no Capítulo 2, incluindo convenções de chamada e alguns truques inteligentes para “linkagem”. A linguagem assembly inclui todas as instruções adequadas do RISC-V, além de algumas instruções úteis que estão fora do RISC-V. Essas *pseudoinstruções*, que são variações inteligentes de instruções reais, facilitam a escrita de programas em assembly sem a necessidade de complicar a ISA.

Os próximos três capítulos descrevem as extensões padrão RISC-V que, quando adicionadas ao RV32I, coletivamente chamamos RV32G (G é para geral):

- Capítulo 4: Multiplicação e Divisão (RV32M)
- Capítulo 5: Ponto Flutuante (RV32F and RV32D)
- Capítulo 6: Instruções Atômicas (RV32A)

O cartão de referência também é chamado de *green card* por causa da sombra da cor de fundo do resumo de uma página das ISAs da década de 1960. Mantivemos o plano de fundo branco para legibilidade em vez de verde para precisão histórica.

O “cartão de referência” do RISC-V nas páginas 3 e 4 é um resumo útil com *todas* instruções RISC-V apresentadas neste livro: RV32G, RV64G e RV32/64V.

O Capítulo 7 descreve a extensão opcional compactada RV32C, um excelente exemplo da elegância do RISC-V. Ao restringir as instruções de 16 bits para serem versões curtas das instruções existentes de 32 bits do RV32G, elas são quase gratuitas. O assembler pode escolher o tamanho da instrução, permitindo que o programador assembly e o compilador sejam indiferentes ao RV32C. O decodificador de hardware para traduzir instruções RV32C de 16 bits em instruções RV32G de 32 bits precisa de apenas 400 portas, o que representa apenas uma pequena porcentagem da implementação mais simples do RISC-V.

O Capítulo 8 introduz o RV32V, a extensão vetorial. As instruções vetoriais são outro exemplo de elegância ISA, em comparação com as numerosas instruções de força bruta *Single Instruction Multiple Data (SIMD)* do ARM-32, MIPS-32 e x86-32. De fato, centenas de instruções adicionadas ao x86-32 na Figura 1.2 eram SIMD, e centenas mais estão surgindo. O RV32V é ainda mais simples que a maioria das ISAs vetoriais, pois associa o tipo de dados e o seu comprimento aos registradores vetoriais, em vez de incorporá-los nos opcodes. O RV32V pode ser o motivo mais convincente para mudar de uma ISA convencional baseada em SIMD para RISC-V.

O Capítulo 9 mostra a versão de endereços de 64 bits do RISC-V, RV64G. Como explica o capítulo, os arquitetos RISC-V precisaram apenas ampliar os registradores e adicionar algumas versões word, doubleword ou long das instruções RV32G para estender o endereço de 32 para 64 bits.

O Capítulo 10 descreve as instruções do sistema, mostrando como o RISC-V lida com paginação e os modos de privilégio Machine, User e Supervisor.

O último capítulo fornece uma descrição rápida das extensões restantes que estão atualmente sendo consideradas pela RISC-V Foundation.

Em seguida vem a maior seção do livro, o Apêndice A, um resumo do conjunto de instruções em ordem alfabética. Esse define a ISA RISC-V completa com todas as extensões mencionadas acima e todas as pseudoinstruções em cerca de 50 páginas, um testemunho da simplicidade do RISC-V.

O Apêndice B mostra operações comuns da linguagem assembly e a quais instruções elas correspondem no RV32I, ARM-32 e x86-32. Essas três figuras são seguidas por um pequeno programa em C e a saída do compilador para três ISAs. Este apêndice tem dois propósitos: Para os leitores já familiarizados com as ISAs ARM-32 ou x86-32, isso é outra maneira interessante de aprender RISC-V. E para ajudar os programadores que estão convertendo programas assembly escritos nessas ISAs mais antigas para o RISC-V.

Nós finalizamos o livro com um índice.



Simplicidade

ISA	Páginas	Palavras	Horas de leitura	Semanas de leitura
RISC-V	236	76,702	6	0.2
ARM-32	2736	895,032	79	1.9
x86-32	2198	2,186,259	182	4.5

**Figura 1.6:** Número de páginas e palavras dos manuais ISA [Waterman and Asanović 2017a], [Waterman and Asanović 2017b], [Intel Corporation 2016], [ARM Ltd. 2014]. Horas e semanas para concluir pressupõem leitura a 200 palavras por minuto durante 40 horas por semana. Baseado em parte na Figura 1 de [Baumann 2017].

## 1.5 Considerações Finais

*It is easy to see by formal-logical methods that there exist certain [instruction sets] that are in abstract adequate to control and cause the execution of any sequence of operations ... The really decisive considerations from the present point of view, in selecting an [instruction set], are more of a practical nature: simplicity of the equipment demanded by the [instruction set], and the clarity of its application to the actually important problems together with the speed of its handling of those problems.*

—[von Neumann et al. 1947, 1947]

O RISC-V é uma ISA recente, iniciada do zero, minimalista e aberta, informada por erros de ISA anteriores. O objetivo dos arquitetos RISC-V é que este seja eficaz para todos os dispositivos de computação, do menor ao mais rápido. Segundo o conselho de von Neumann de 70 anos atrás, esta ISA enfatiza a simplicidade para manter os custos baixos e ter muitos registradores e velocidade de instrução transparente para ajudar compiladores e programadores de linguagem assembly a mapear problemas realmente importantes para um código rápido e apropriado.

Um indicativo de complexidade é o tamanho da documentação. A Figura 1.6 mostra o tamanho dos manuais do conjunto de instruções para RISC-V, ARM-32 e x86-32 medidos em páginas e palavras. Se você ler manuais de ISAs como um trabalho de período integral—8 horas por dia durante 5 dias por semana—levaria meio mês para ler uma única passagem pelo manual do ARM-32 e um mês completo para o x86-32. Neste nível de complexidade, talvez nenhuma pessoa compreenda inteiramente o ARM-32 ou o x86-32. Usando esta métrica de senso comum, RISC-V é  $\frac{1}{12}$  da complexidade do ARM-32 e  $\frac{1}{10}$  a  $\frac{1}{30}$  a complexidade do x86-32. De fato, o resumo do ISA RISC-V incluindo todas as extensões é de apenas duas páginas (veja o Cartão de Referência).

Esta ISA mínima e aberta foi revelada em 2011 e agora é apoiado por uma fundação que irá evoluí-la adicionando extensões opcionais estritamente baseadas em justificativas técnicas após um debate prolongado. A abertura permite implementações gratuitas e compartilhadas do RISC-V, o que reduz os custos e as chances de segredos maliciosos indesejados serem escondidos em um processador.

No entanto, o hardware sozinho não faz um sistema. Os custos de desenvolvimento de software provavelmente superam os custos de desenvolvimento de hardware, portanto, embora o hardware estável seja importante, o software estável é mais importante. Ele requer sistemas operacionais, carregadores de inicialização, software de referência e ferramentas de software populares. A base oferece estabilidade para o ISA geral, e a base congelada significa que o núcleo RV32I que é o alvo da pilha de software nunca será alterado. Por sua ampla adoção e abertura, o RISC-V pode desafiar o domínio das ISAs proprietárias.

**Uma versão anterior do relatório bem escrito de John von Neumann** foi tão influente que esse estilo de computador é comumente chamado de arquitetura de *von Neumann*, embora este relatório tenha sido baseado no trabalho de outros. Foi escrito três anos antes do primeiro computador de programa armazenado em memória estar operacional!



Simplicidade



Elegância

Elegância é uma palavra que é raramente aplicada a ISAs, mas depois de ler este livro, você pode concordar conosco que se aplica ao RISC-V. Vamos destacar características que acreditamos que indicam elegância com um ícone da Mona Lisa nas margens.

## 1.6 Para Saber Mais

ARM Ltd. ARM Architecture Reference Manual: ARMv7-A and ARMv7-R Edition, 2014. URL <http://infocenter.arm.com/help/topic/com.arm.doc.ddi0406c/>.

A. Baumann. Hardware is the new software. In *Proceedings of the 16th Workshop on Hot Topics in Operating Systems*, pages 132–137. ACM, 2017.

C. Celio, D. Patterson, and K. Asanovic. The Berkeley Out-of-Order Machine (BOOM): an industry-competitive, synthesizable, parameterized RISC-V processor. *Tech. Rep. UCB/EECS-2015-167, EECS Department, University of California, Berkeley*, 2015.

S. Gal-On and M. Levy. Exploring CoreMark - a benchmark maximizing simplicity and efficacy. *The Embedded Microprocessor Benchmark Consortium*, 2012.

Intel Corporation. *Intel 64 and IA-32 Architectures Software Developer's Manual, Volume 2: Instruction Set Reference*. September 2016.

S. P. Morse. The Intel 8086 chip and the future of microprocessor design. *Computer*, 50(4): 8–9, 2017.

D. A. Patterson and J. L. Hennessy. *Computer Organization and Design RISC-V Edition: The Hardware Software Interface*. Morgan Kaufmann, 2017.

S. Rodgers and R. Uhlig. X86: Approaching 40 and still going strong, 2017.

J. L. von Neumann, A. W. Burks, and H. H. Goldstine. Preliminary discussion of the logical design of an electronic computing instrument. *Report to the U.S. Army Ordnance Department*, 1947.

A. Waterman. *Design of the RISC-V Instruction Set Architecture*. PhD thesis, EECS Department, University of California, Berkeley, Jan 2016. URL <http://www2.eecs.berkeley.edu/Pubs/TechRpts/2016/EECS-2016-1.html>.

A. Waterman and K. Asanović, editors. *The RISC-V Instruction Set Manual Volume II: Privileged Architecture Version 1.10*. May 2017a. URL <https://riscv.org/specifications/privileged-isa/>.

A. Waterman and K. Asanović, editors. *The RISC-V Instruction Set Manual, Volume I: User-Level ISA, Version 2.2*. May 2017b. URL <https://riscv.org/specifications/>.