ORGANIZAÇÃO E ARQUITETURA DE COMPUTADORES

Arquitetura RISC-V

Profª. Fabiana F F Peres

Apoio: Camile Bordini

Conjunto de instruções

Você pode pensar que as linguagens dos computadores são tão diversas quanto as das pessoas, mas, na realidade, as linguagens dos computadores são bastante semelhantes, mais parecidas com dialetos regionais do que com línguas independentes. Assim, após aprender uma linguagem, é fácil aprender outras.

Patterson & Henessy

Conjunto de instruções

- **MIPS** é um exemplo de conjuntos de instruções projetado na década de 1980. O **RISC-V** segue um design semelhante.

- O **Intel x86** teve origem na década de 1970, mas ainda hoje está presente tanto em PC's quanto na nuvem da era pós-PC.

Conjunto de instruções

- Esta semelhança ocorre pois:
  - Todos os computadores são construídos a partir de tecnologias de hardware semelhantes
  - Há operações básicas que todos os computadores devem fornecer.
  - Os projetistas de computadores querem encontrar uma linguagem que facilite a construção do hardware e do compilador, ao mesmo tempo que maximiza o desempenho e minimiza custos e energia

RISC-V

RISC-V

- Projeto iniciou em 2010 na Universidade da Califórnia (Berkeley)
- Não é uma arquitetura proprietária, como a MIPS, ARM ou X86
- Permite qualquer pessoa ou empresa projetar e vender chips e softwares RISC-V sem pagar *royalties*
- Pertence a uma fundação aberta, sem fins lucrativos; a *RISCV Foundation*
  - www.riscv.org

RISC-V

- Tem como objetivo tornar-se uma **ISA Universal** e para isso, deve, por exemplo:
  - Atender a todos os tamanhos de processadores (de embarcados até os de alto desempenho)
  - Funcionar bem com uma grande variedade de softwares e linguagens de programação populares.
  - Acomodar todas as tecnologias de implementação: chips customizados e futuras tecnologias de dispositivos.
  - Etc.

Fundadores

- Krste Asanović
- David Patterson
- DARPA (financiamento inicial)

Apoiadores

AMD, Andes Technology, BAE Systems, Berkeley Architecture Research, Bluespec, Inc., Cortus, **Google**, GreenWaves Technologies, Hewlett Packard Enterprise, **Huawei**, **IBM**, Imperas Software, ICT, IIT Madras, Lattice Semiconductor, Mellanox Technologies, Microsemi, Micron, **Nvidia**, NXP, **Oracle**, **Qualcomm**, Rambus Cryptography Research, Western Digital, e SiFive.

Apoiadores

>$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		

Princípios RISC

I. *"Simplicidade favorece a regularidade"*

II. *"Menor é mais rápido"*

III. *"Bons projetos demandam bons compromissos"*

IV. *"Faça o caso comum mais rápido"*


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
de 64 bits.
acima, o texto mostra como o RISC-V
passadas, evitando os erros cometidos.
Guia Prático RISC-V
avid Patterson
rew Waterman
ICO
RISC–V
UITETURA ABERTA
COMPUTER
ORGANIZATION
AND DESIGN
THE HARDWARE/SOFTWARE INTERFAC
RISC-V EDITION
DAVID A. PATTERSON
JOHN L. HENNESSY
COMPUTER ORGANIZATION
AND DESIGN RISC-V EDITION
THE HARDWARE SOFTWARE INTERFACE
SECOND EDITION
DAVID A. PATTERSON & JOHN L. HENNESSY

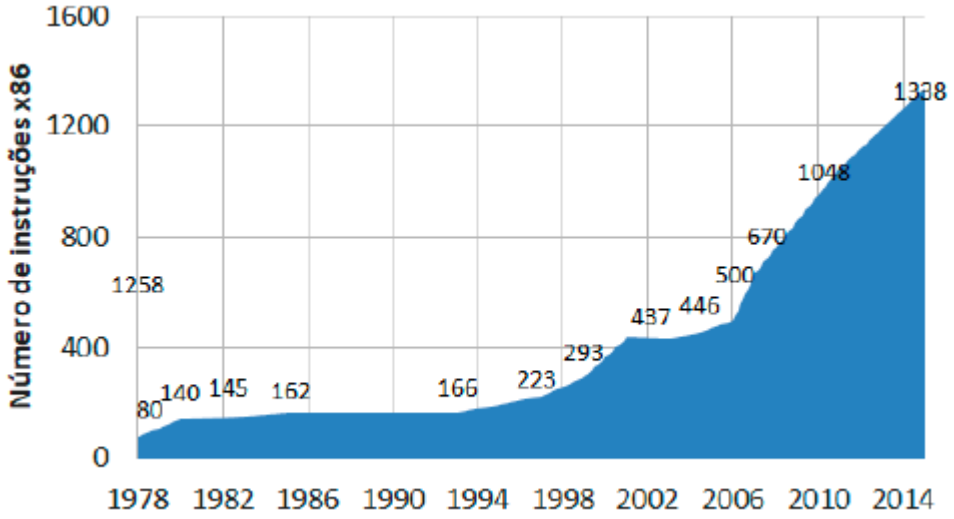
ISAs Incrementais

Utilizado na arquitetura CISC

- Novos processadores implementam novas extensões ISA e todas as extensões do passado.
- Preocupa-se em manter a compatibilidade para que versões binárias de programas de décadas possam ser executadas corretamente no processador mais recente.

ISAs Incrementais

- Crescimento no número de instruções para o 80x86 de 1978 - adicionou aproximadamente três instruções por mês ao longo de sua vida útil.
  - Toda implementação do x86-32 implementa os erros das extensões anteriores, mesmo quando elas não fazem mais sentido.



ISAs Modulares

Utilizado na arquitetura RISC

- Possui um núcleo básico:
  - No **RISC-V** é chamado de **RV32I**
  - Não deve ser alterado (núcleo estável)
    - Favorece principalmente os criadores de compiladores, desenvolvedores de SO e programadores assembly
- Possui extensões opcionais que o hardware pode incluir ou não, dependendo das necessidades
- Permite implementações muito pequenas e de baixo consumo de energia

ISAs Modulares

- Convenção:
  - anexar letras de extensão ao nome para indicar quais estão incluídas na versão.
  - Exemplo: **RV32IMFD** adiciona:
    - Instruções de base obrigatórias: **RV32I**
    - Extensão de multiplicação de ponto flutuante: **RV32M**
    - Extensão de ponto flutuante de precisão simples: **RV32F**
    - Extensão de ponto flutuante de precisão dupla: **RV32D**

Conceitos: Arquitetura

Comportamento funcional de um sistema computacional, do ponto de vista do programador

"Conjunto de recursos percebidos pelo programador em linguagem de máquina, tais como

- *Registradores*
- *Tipos de dados manipulados pelas instruções*
- *Organização da memória principal*
- *Modos de endereçamento*
- *Conjunto de instruções ..."*

Conceitos: Arquitetura

"Conjunto de recursos percebidos pelo programador em linguagem de máquina, tais como

1. *Registradores*
2. *Tipos de dados manipulados pelas instruções*
3. *Organização da memória principal*
4. *Modos de endereçamento*
5. *Conjunto de instruções ..."*

Conceitos: Arquitetura

"Conjunto de recursos percebidos pelo programador em linguagem de máquina, tais como

1. ***Registradores***
2. *Tipos de dados manipulados pelas instruções*
3. *Organização da memória principal*
4. *Modos de endereçamento*
5. *Conjunto de instruções ..."*

Registradores RV32I

- Possui um banco de **32 registradores**: **x0** à **x31** (convenção)
- Cada registrador tem **32 bits**

31 0

x0 / zero	Zero hardwired
x1 / ra	Endereço de retorno
x2 / sp	Ponteiro de pilha
x3 / gp	Ponteiro global
x4 / tp	Ponteiro de thread
x5 / t0	Temporário
x6 / t1	Temporário
x7 / t2	Temporário
x8 / s0 / fp	Registrador salvo, ponteiro de quadro
x9 / s1	Registrador salvo
x10 / a0	Argumento de função, valor de retorno
x11 / a1	Argumento de função, valor de retorno
x12 / a2	Argumento de função
x13 / a3	Argumento de função
x14 / a4	Argumento de função
x15 / a5	Argumento de função
x16 / a6	Argumento de função
x17 / a7	Argumento de função

x18 / s2	Registrador salvo
x19 / s3	Registrador salvo
x20 / s4	Registrador salvo
x21 / s5	Registrador salvo
x22 / s6	Registrador salvo
x23 / s7	Registrador salvo
x24 / s8	Registrador salvo
x25 / s9	Registrador salvo
x26 / s10	Registrador salvo
x27 / s11	Registrador salvo
x28 / t3	Temporário
x29 / t4	Temporário
x30 / t5	Temporário
x31 / t6	Temporário

32

31 0

pc

32

Registradores RV32I

Name	Register number	Usage	Preserved on call?
x0	0	The constant value 0	n.a.
x1 (ra)	1	Return address (link register)	yes
x2 (sp)	2	Stack pointer	yes
x3 (gp)	3	Global pointer	yes
x4 (tp)	4	Thread pointer	yes
x5-x7	5–7	Temporaries	no
x8-x9	8–9	Saved	yes
x10-x17	10–17	Arguments/results	no
x18-x27	18–27	Saved	yes
x28-x31	28–31	Temporaries	no

Registradores RV32I

- Grupos de 32 bits ocorrem com tanta frequência que recebem o nome de "**palavra**" (***word***) em RISC-V
- Outro tamanho comum: grupo de 64 bits (**palavra dupla**)

- Motivo para o limite de 32 registradores:
  - Muitos registradores faria aumentar o tempo do ciclo do clock, simplesmente porque os sinais eletrônicos levam mais tempo quando eles precisam viajar mais longe.
  - O número de bits que seriam necessários no formato da instrução (próximos slides)

"Menor é mais rápido"

Conceitos: Arquitetura

"Conjunto de recursos percebidos pelo programador em linguagem de máquina, tais como

1. *Registradores*
2. ***Tipos de dados manipulados pelas instruções***
3. *Organização da memória principal*
4. *Modos de endereçamento*
5. *Conjunto de instruções ..."*

Sistema de numeração

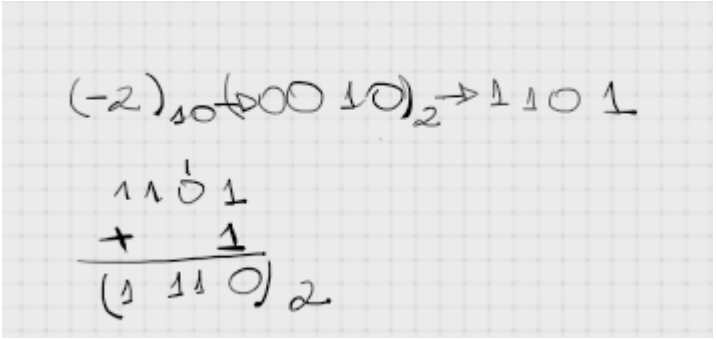
a) Números inteiros:

– *Sem sinal*: **binário puro**

– *Com sinal:* **complemento de 2**

- Obtém a representação do valor sem o sinal
- Inverte os bits (complemento de 1)
- Adiciona 1

– Exemplo:



Sistema de numeração

b) Números reais: **Padrão IEEE 754**

- Simplifica a troca de dados, algoritmos e aumenta a precisão dos números
- Utilizado praticamente por todos os computadores desde 1980
- Três passos para transformar um número decimal em binário:
  1. Representação do número em binário (parte inteira e parte fracionária)
  2. Normalização
  3. Enquadramento

Sistema de numeração

- Números reais: **Padrão IEEE 754 (precisão simples)**

S	E(8 bits)	M (23 bits)

  - **S (sinal)**
    - 0 para número positivo
    - 1 para número negativo
  - **E (expoente)**: Excesso de $2^{N-1} - 1$ (Intervalo: 0 a 255)
  - **M (mantissa)**: acomoda o significando do número obtido, em notação científica normalizada

Padrão IEEE 754 - conversão

- Três passos para transformar um número decimal em binário via IEEE 754
  1. Representação do número em binário
     - Converter a parte inteira e a parte fracionária
  2. Normalização
     - Representar o número convertido na notação científica normalizada

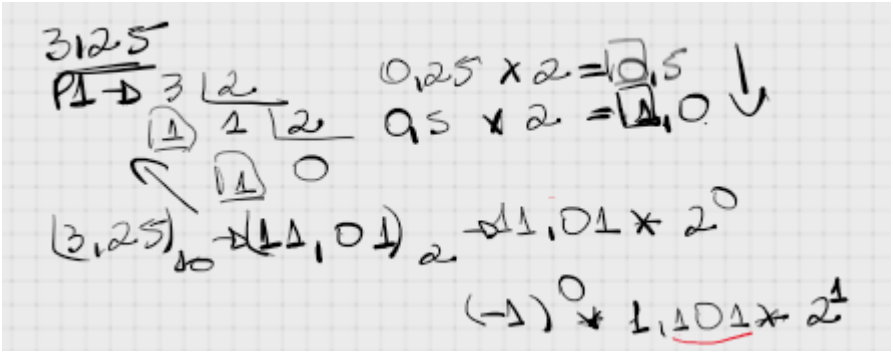
$$(-1)^S * (1. + fração) * 2^{(E-bias)}$$

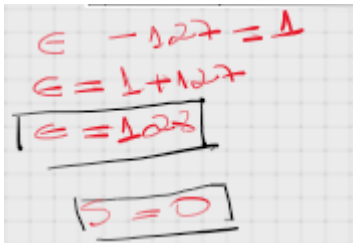
  3. Enquadramento:
     - Distribuir as informações (sinal, expoente e mantissa) nos 32 bits

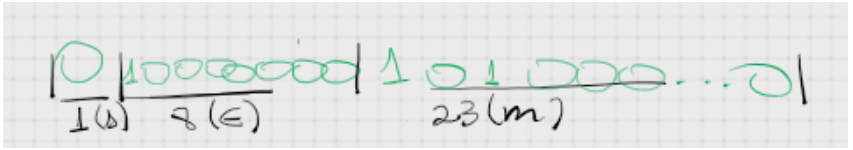
Padrão IEEE 754

- Ex: Mostre a representação binária para o número a seguir em precisão simples

a) 3,25	d) 0,5
b) -0,75	e) -2,5
c) 0,08	f) -9,6







Sistema de numeração (cont.)

"Bons projetos demandam bons compromissos"

- Compromisso entre o tamanho da mantissa e do expoente
  - Aumentar o tamanho da *mantissa*: melhora a precisão
  - Aumentar o tamanho do *expoente*: aumenta o intervalo de números que podem ser representados

Padrão IEEE 754

- O valor zero recebe o valor reservado ‘000…0’ para seu expoente e o hardware não acrescenta o 1 inicial
- Utiliza símbolos especiais para representar eventos incomuns
  - Divisão por 0
    - Resultado é representado em um padrão de bits que signifique $+\infty$ ou $-\infty$
    - O maior expoente é reservado para estes casos
  - Operações inválidas
    - 0/0, subtração de infinito por infinito, …
    - *NaN*

Padrão IEEE 754

Single precision		Object represented
Exponent	Fraction	
0	0	0
0	nonzero	± denormalized number
1–254	anything	± floating-point number
255	0	± infinity
255	nonzero	NaN (Not a Number)

Conceitos: Arquitetura

"Conjunto de recursos percebidos pelo programador em linguagem de máquina, tais como

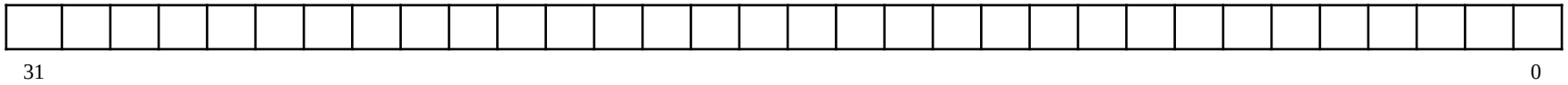
1. *Registradores*
2. *Tipos de dados manipulados pelas instruções*
3. *Organização da memória principal*
4. *Modos de endereçamento*
5. ***Conjunto de instruções** ..."*

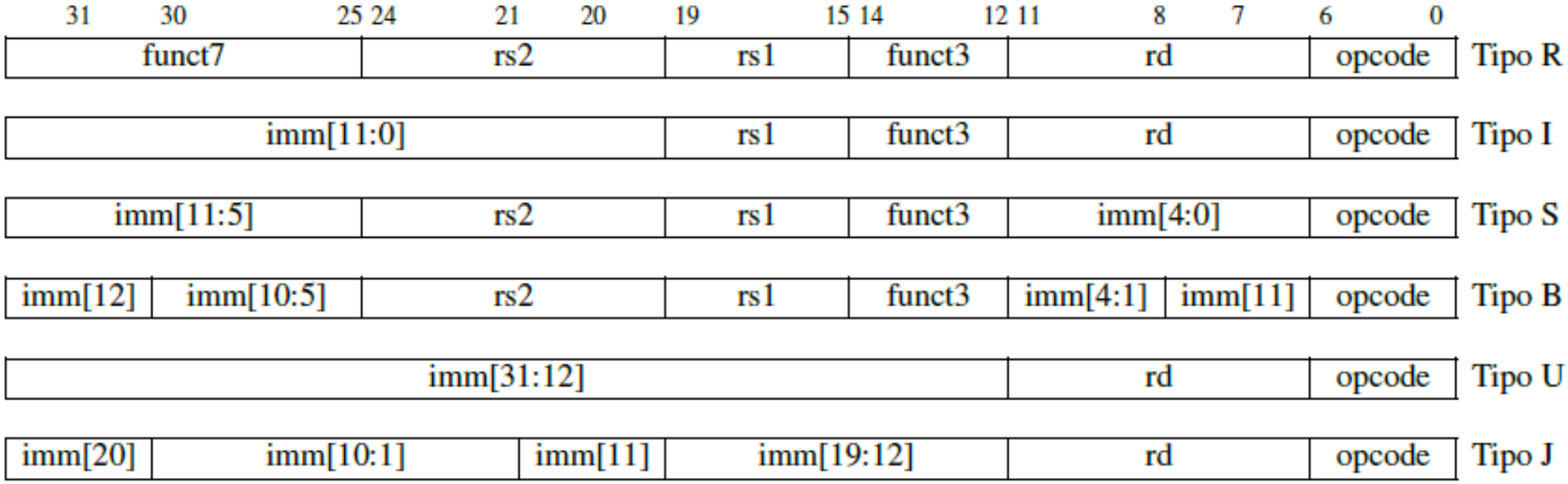
Formato das instruções

- O conjunto de instruções é o vocabulário de comandos compreendido por uma determinada arquitetura

- Organizadas em **seis classes**
  1. **Tipo-R** para operações de registradores
  2. **Tipo-I** para valores imediatos short e loads
  3. **Tipo-S** para stores
  4. **Tipo-B** para desvios condicionais
  5. **Tipo-U** para valores imediatos longos
  6. **Tipo-J** para saltos incondicionais

Formato das instruções





Instruções do Formato R

- Para instruções lógicas e aritméticas

- add x5, x6, x7 // x5 = x6 + x7 //soma
- sub x5, x6, x7 // x5 = x6 - x7 //subtração
- and x5, x6, x7 // x5 = x6 & x7 //and lógico
- or x5, x6, x7 // x5 = x6 | x7 //or lógico

Instruções do Formato R

- Este tipo de notação é rígido no RISC-V porque cada **instrução aritmética** executa **apenas uma operação** e deve sempre ter **exatamente três variáveis.**

- Exemplo: somar quatro variáveis **b**, **c**, **d**, **e** na variável **a**.
  - add a, b, c *//The sum of b and c is placed in a*
  - add a, a, d *//The sum of b, c, and d is now in a*
  - add a, a, e *//The sum of b, c, d, and e is now in a*

Instruções do Formato R

- No exemplo anterior:
  - *necessário 3 instruções para somar 4 variáveis*
  - *cada linha pode ter no máximo 1 instrução*

- Além disso, as exigências de que:
  - ... uma operação por instrução RISC-V; e
  - ... cada instrução deve ter exatamente três operandos, (nem mais nem menos)
- mantém o hardware simples: "*Simplicidade favorece a regularidade*"

Exemplo 1

- Este trecho de um programa C contém cinco variáveis a, b, c, d, e.

$$a = b + c$$
$$d = a - e$$

- Código assembly: ?

Exemplo 1

- Este trecho de um programa C contém cinco variáveis a, b, c, d, e.

$$a = b + c$$
$$d = a - e$$

- Código assembly:

```
add a, b, c
sub d, a, e
```

Exemplo 2

- Este trecho de um programa C contém cinco variáveis f, g, h, i, j.

$$f = (g+h) - (i+j)$$

  - considere que as variáveis f, g, h, i e j estão armazenadas nos registradores x19, x20, x21, x22 e x23 respectivamente

- Código Assembly: ?

Obs: é tarefa do compilador associar as variáveis dos programas aos registradores.

Exemplo 2

- Este trecho de um programa C contém cinco variáveis f, g, h, i, j.

$$f = (g+h) - (i+j)$$

  – considere que as variáveis f, g, h, i e j estão armazenadas nos registradores x19, x20, x21, x22 e x23 respectivamente

- Código Assembly:

```
add x5, x20, x21     //x5 = g + h
add x6, x22, x23     //x6 = i + j
sub x19, x5, x6      //f = x5 - x6
```

Formato R

funct7	rs2	rs1	funct3	rd	opcode
7 bits	5 bits	5 bits	3 bits	5 bits	7 bits

- **opcode**: operação da instrução
- **rd:** registrador destino. Armazena o resultado da operação
- **funct3:** opcode adicional
- **rs1:** primeiro operando
- **rs2:** segundo operando
- **funct7:** opcode adicional

Instruction	Format	funct7	rs2	rs1	funct3	rd	opcode
add (add)	R	0000000	reg	reg	000	reg	0110011
sub (sub)	R	0100000	reg	reg	000	reg	0110011

Exemplo

```
add x9, x20, x21
```

0000000	10101	10100	000	01001	0110011
7 bits	5 bits	5 bits	3 bits	5 bits	7 bits

0	21	20	0	9	51

funct7	rs2	rs1	funct3	rd	opcode
7 bits	5 bits	5 bits	3 bits	5 bits	7 bits

Exemplo 2

- Código C:
  - `f= (g+h) - (i+j);`
- Código Assembly:

```
1. add x5, x20, x21     //x5 = g + h
2. add x6, x22, x23     //x6 = i + j
3. sub x19, x5, x6      //f = x5 - x6
```

- Preencha o quadro das 3 instruções acima:
  - Em decimal
  - Em binário

Exemplo 2

- Em decimal:

	funct7	**rs2**	**rs1**	**funct3**	**rd**	**opcode**
1	*0*	*21*	*20*	*0*	*5*	*51*
2	*0*	*23*	*22*	*0*	*6*	*51*
3	*32*	*6*	*5*	*0*	*19*	*51*

- Em binário

	funct7	**rs2**	**rs1**	**funct3**	**rd**	**opcode**
1	*0000000*	*10101*	*10100*	*000*	*00101*	*0110011*
2	*0000000*	*10111*	*10110*	*000*	*00110*	*0110011*
3	*0100000*	*00110*	*00101*	*000*	*10011*	*0110011*

Referências

- PATTERSON, David A; HENNESSY, John L; Computer Organization and Design – The hardware/software interface RISC-V edition; Elsevier – Morgan Kaufmann/ Amsterdam.
- PATTERSON, David; Waterman, Andrew; The RISC-V reader: an open architecture atlas; First edition. Berkeley, California: Strawberry Canyon LLC, 2017.