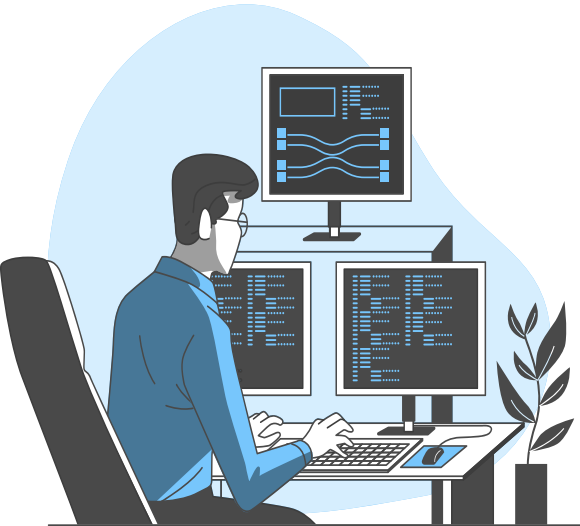


SSC0510 – Arquitetura de Computadores RISC – V



Grupo 2:

Enzo Bustamante, 9863437

Gabriel Fontes, 10856803

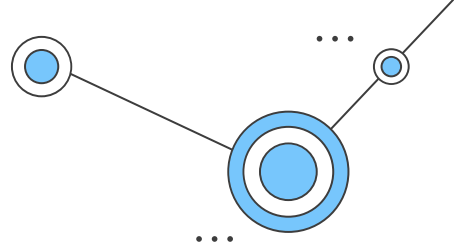
Giovanna Fardini, 10260671

Thales Damasceno, 11816150

Vinicius Baca, 10788589



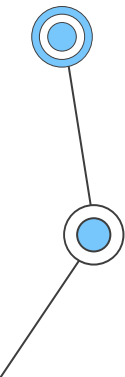
Introdução



- Arquitetura RISC - *load/store*;
- *Open Standard Instruction Set Architecture (ISA)* - menor risco de obsolescência, permite intercambiação em *hardware* e *software*;
- Licenças *open source* permissivas;
- Instruções divididas em: padrão (não podem ter conflito entre si) e não padrão (extensões que podem ser especializadas e conflitarem entre si).

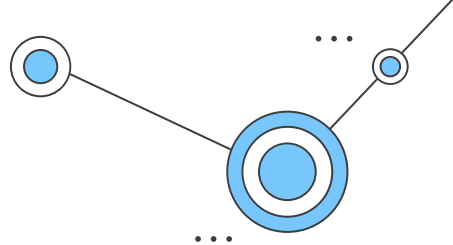


RISC-V: The Free and Open RISC
Instruction Set Architecture

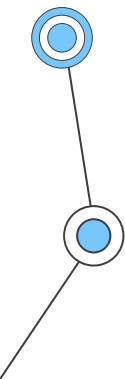


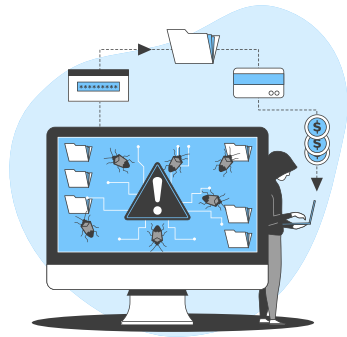


Introdução

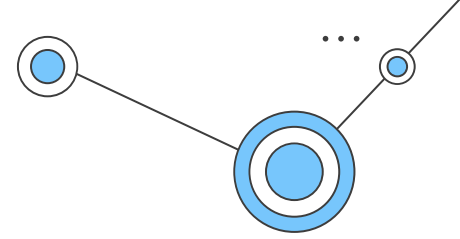


- Projetado para suportar customização e especialização
 - Instruções podem ser estendidas pela adição de um ou mais conjuntos, mas instruções nativas não podem ser redefinidas;
- Instruções de base inteira de 32 bits e reconhecimento de instruções de tamanho variável múltiplas de 16 bits.
 - Duas variações - RV32I e RV64I (e desenvolvimento de uma versão de 128 bits ainda em estudo).





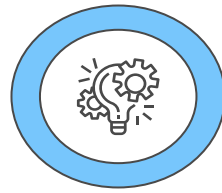
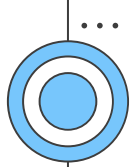
Introdução



xxxxxxxxxxxxxxxxaa		16-bit (aa ≠ 11)	
xxxxxxxxxxxxxxxxxxxx	xxxxxxxxxxxxbbb11	32-bit (bbb ≠ 111)	
···xxxx	xxxxxxxxxxxxxxxxxxxx	xxxxxxxxxxx011111	48-bit
···xxxx	xxxxxxxxxxxxxxxxxxxx	xxxxxxxxxxx0111111	64-bit
···xxxx	xxxxxxxxxxxxxxxxxxxx	xnnnxxxxx1111111	(80+16*nnn)-bit, nnn≠111
···xxxx	xxxxxxxxxxxxxxxxxxxx	x111xxxxx1111111	Reserved for ≥192-bits

Byte Address: base+4 base+2 base

Figure 1.1: RISC-V instruction length encoding.



Ordenação de bytes

- Sistema de memória *little endian* (mais comum no mercado e natural para desenvolvedores);
- Possibilidade em aberto de adição de estrutura não padrão *big endian*

```
// Store 32-bit instruction in x2 register to location pointed to by x3.  
sh    x2, 0(x3)    // Store low bits of instruction in first parcel.  
srli  x2, x2, 16    // Move high bits down to low bits, overwriting x2.  
sh    x2, 2(x3)    // Store high bits in second parcel.
```

Instruções

- 4 formatos de instruções atômicos (R, I, S, U);
- Mantém os registradores fonte e destino sempre na mesma posição e ordem ...
no código de todas as instruções.

31	25 24	20 19	15 14	12 11	7 6	0	
funct7		rs2	rs1	funct3	rd	opcode	R-type
imm[11:0]			rs1	funct3	rd	opcode	I-type
imm[11:5]		rs2	rs1	funct3	imm[4:0]	opcode	S-type
imm[31:12]					rd	opcode	U-type



Instruções de codificação imediata



- +2 variações no formato de instruções (B, J) para tratar codificação imediata;
- B é similar à S, exceto que os bits do meio ficam em posições fixas e o bit de menor ordem no formato S corresponderá ao bit de maior ordem no formato B;
- U é similar à J, exceto que em U os 20 bits reservados são shiftados 12 bits para à esquerda enquanto que em J são shiftados somente 1 bit à esquerda

Instruções de codificação imediata

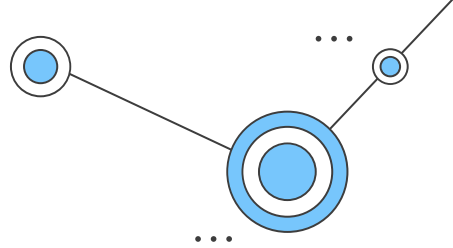
32-bit RISC-V instruction formats

Format	Bit																																	
	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0		
Register/register	funct7							rs2					rs1					funct3			rd					opcode								
Immediate	imm[11:0]												rs1					funct3			rd					opcode								
Upper immediate	imm[31:12]																				rd					opcode								
Store	imm[11:5]							rs2					rs1					funct3			imm[4:0]					opcode								
Branch	[12]	imm[10:5]						rs2					rs1					funct3			imm[4:1]			[11]	opcode									
Jump	[20]	imm[10:1]										[11]	imm[19:12]										rd					opcode						

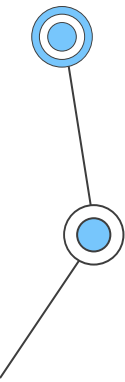
- *opcode* (7 bits): Partially specifies which of the 6 types of *instruction formats*.
- *funct7*, and *funct3* (10 bits): These two fields, further than the *opcode* field, specify the operation to be performed.
- *rs1*, *rs2*, or *rd* (5 bits): Specifies, by index, the register, resp., containing the first operand (i.e., source register), second operand, and destination register to which the computation result will be directed.



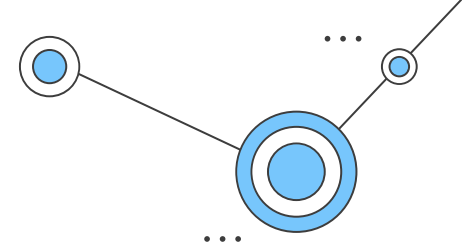
Instruções de computação de inteiros



- Codificadas como:
- registrador-imediata: usa o tipo I - ADDI, SLTI (set less than immediate), SLTIU para tipo *unsigned*, ANDI, ORI, XORI, instruções de *shift* por uma constante.
 - registrador-registrador: usa o tipo R - ADD, SUB, AND, OR, XOR, SLT, SLTU, SLL, SRL, SRA e NOP.



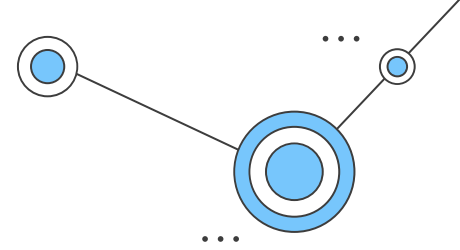
Instruções de computação de inteiros



Registrador-imediata:

31	20 19	15 14	12 11	7 6	0
imm[11:0]	rs1	funct3	rd	opcode	
12	5	3	5	7	
I-immediate[11:0]	src	ADDI/SLTI[U]	dest	OP-IMM	
I-immediate[11:0]	src	ANDI/ORI/XORI	dest	OP-IMM	
imm[11:5]	imm[4:0]	rs1	funct3	rd	opcode
7	5	5	3	5	7
0000000	shamt[4:0]	src	SLLI	dest	OP-IMM
0000000	shamt[4:0]	src	SRLI	dest	OP-IMM
0100000	shamt[4:0]	src	SRAI	dest	OP-IMM

Instruções de computação de inteiros



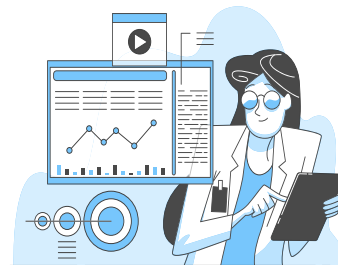
Registrador-registrador:

31	25 24	20 19	15 14	12 11	7 6	0
funct7	rs2	rs1	funct3	rd	opcode	
7	5	5	3	5	7	
0000000	src2	src1	ADD/SLT/SLTU	dest	OP	
0000000	src2	src1	AND/OR/XOR	dest	OP	
0000000	src2	src1	SLL/SRL	dest	OP	
0100000	src2	src1	SUB/SRA	dest	OP	

imm[11:0]	rs1	funct3	rd	opcode
12	5	3	5	7
0	0	ADDI	0	OP-IMM

Instruções de transferência de controle

- 2 tipos:
 - *Jumps* incondicionais
 - JAL (jump and link - tipo J), JALR (jump and link register - tipo I),
 - Desvios condicionais (sempre são do tipo B).
 - BEQ e BNE (igual ou diferente, respectivamente), BLT e BLTU (menor que), BGE e BGEU (maior que).



Instruções de transferência de controle

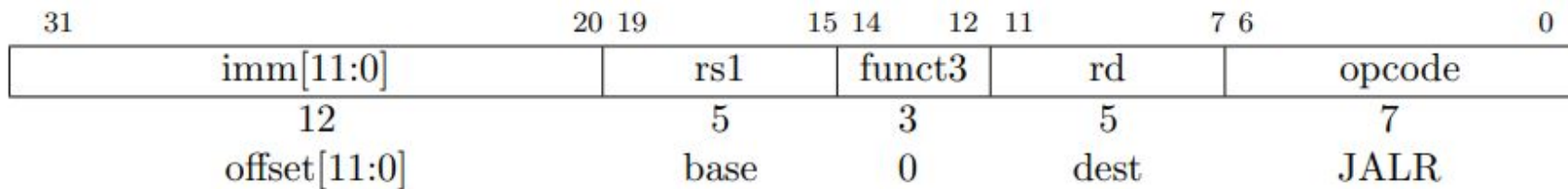
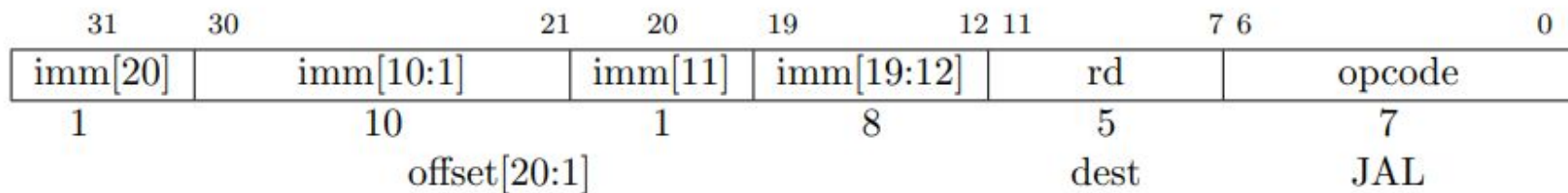
➤ *Jumps* incondicionais: usam endereçamento relativo de PC como suporte a código de posição independente.

- Pilhas de previsão de endereço de retorno: codificadas implicitamente via o número dos registradores usados.
 - JAL só dá um *push* quando $rd = x1/x5$.
 - JALR dá *push/pop* de acordo com a tabela abaixo.

<i>rd</i>	<i>rs1</i>	<i>rs1=rd</i>	RAS action
<i>!link</i>	<i>!link</i>	-	none
<i>!link</i>	<i>link</i>	-	pop
<i>link</i>	<i>!link</i>	-	push
<i>link</i>	<i>link</i>	0	push and pop
<i>link</i>	<i>link</i>	1	push

Instruções de transferência de controle

Jumps incondicionais:

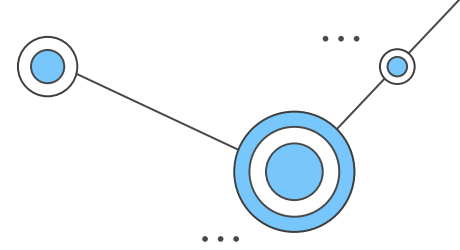


Instruções de transferência de controle

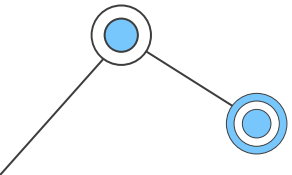
Desvios condicionais:

31	30	25 24	20 19	15 14	12 11	8	7	6	0
imm[12]	imm[10:5]	rs2	rs1	funct3	imm[4:1]	imm[11]	opcode		
1	6	5	5	3	4	1	7		
offset[12,10:5]		src2	src1	BEQ/BNE	offset[11,4:1]		BRANCH		
offset[12,10:5]		src2	src1	BLT[U]	offset[11,4:1]		BRANCH		
offset[12,10:5]		src2	src1	BGE[U]	offset[11,4:1]		BRANCH		

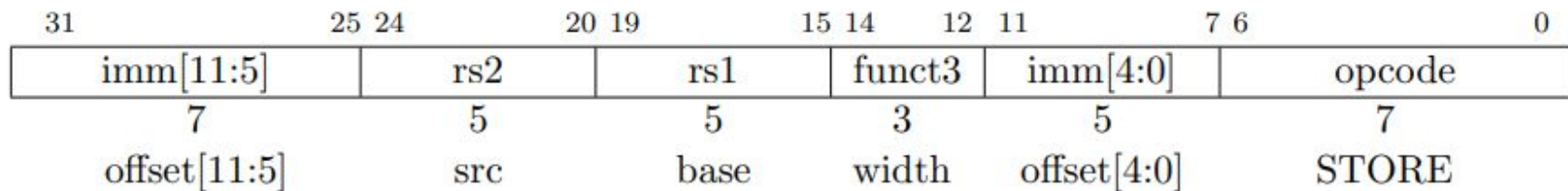
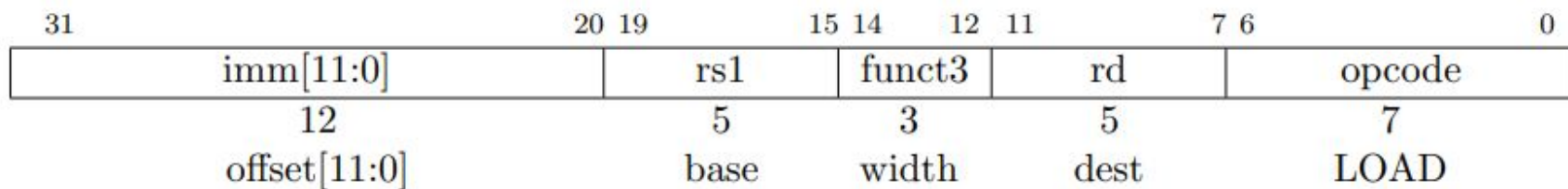
Instruções de leitura e escrita na memória



- Somente as instruções de *load/store* tem permissão de acessar a memória;
- O RV32I libera um espaço de memória ao usuário de 32 bits que é *byte-addressed* e *little-endian*;
- *Load* é do tipo I e *store* é do tipo S.
- As instruções devem estar alinhadas para cada tipo de dados, a base ISA fornece suporte para acessos desalinhados, mas existe perda de performance.

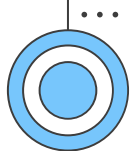


Instruções de leitura e escrita na memória



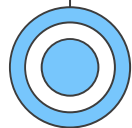
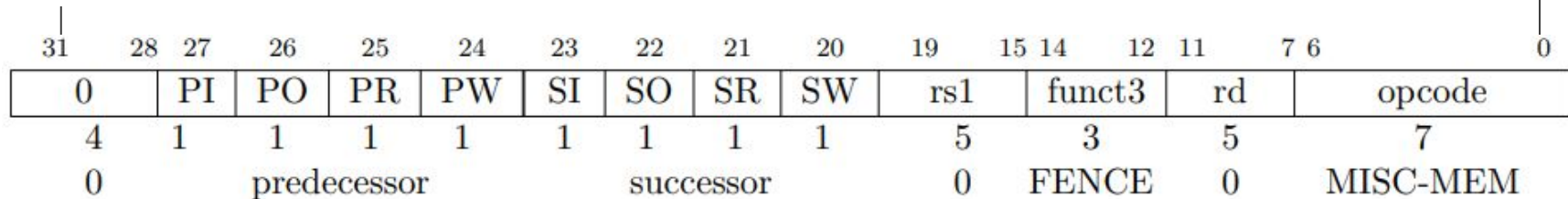
Modelo de memória

- Modelo de memória relaxado - mais compatível com futuras extensões e maior performance com implementações mais simples;
- RISC-V permite a execução de *threads* (*hart*) concorrentes em um único espaço de endereço de usuário;
- Cada *thread* tem seu próprio PC, registrador de estado e executa uma sequência de instruções independentes;



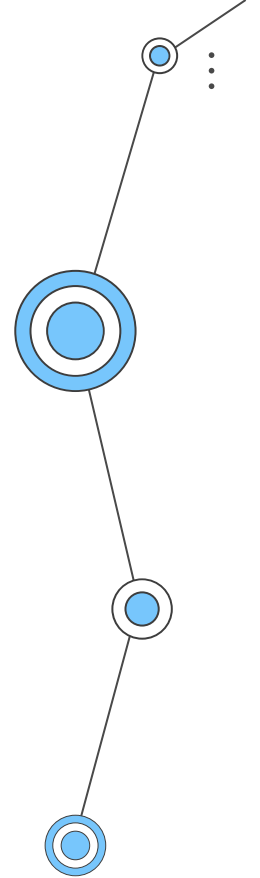
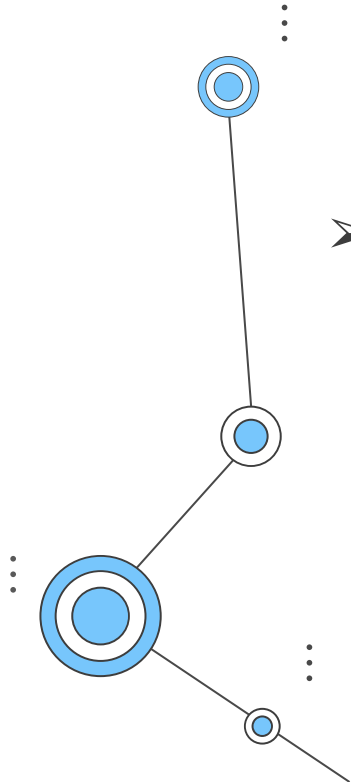
Modelo de memória

- As *threads* podem se comunicar por chamadas ou por sistemas de memória compartilhada;
- Tem uma instrução 'FENCE' para garantir que operações na memória de diferentes *threads* ocorram na ordem correta.



Instruções de controle e estado

- Usadas para acessar funcionalidades que podem precisar de acesso autorizado e são codificadas como tipo I;
- 2 classes:
 - CSRs - controle de leitura-edição-escrita atômica de registradores de controle e estado.
 - Outras instruções privilegiadas.



Instruções de controle e estado

- CSRRW - troca os valores nos CSRs e registradores de inteiros;
- CSRRS - lê, seta os bits nos CSRs e salva em um registrador de inteiros;
- CSRRC - lê e limpa os bits nos CSRs.
- CSRWI/CSRRSI/CSRRCI - análogas, mas obtém os valores a partir de imediatos;

31	20 19	15 14	12 11	7 6	0
csr	rs1	funct3	rd	opcode	
12	5	3	5	7	
source/dest	source	CSRRW	dest	SYSTEM	
source/dest	source	CSRRS	dest	SYSTEM	
source/dest	source	CSRRC	dest	SYSTEM	
source/dest	uimm[4:0]	CSRRWI	dest	SYSTEM	
source/dest	uimm[4:0]	CSRRSI	dest	SYSTEM	
source/dest	uimm[4:0]	CSRRCI	dest	SYSTEM	

Instruções de chamada e interrupção

- ECALL - usada para fazer uma requisição ao SO;
- EBREAK - usada por debuggers.

31	20 19	15 14	12 11	7 6	0
funct12	rs1	funct3	rd	opcode	
12	5	3	5	7	
ECALL	0	PRIV	0	SYSTEM	
EBREAK	0	PRIV	0	SYSTEM	



Outras versões do RISC-V

- RV32E - versão reduzida do RV32I para sistemas embarcados (reduz o número de registradores para 16 e remove os contadores;
- RV64I - suporta espaço de endereçamento de 64 bits;
- RV128I (em desenvolvimento) - suporta espaço de endereçamento de 128 bits, sendo uma extrapolação dos RV32I e RV64I.



Placas Disponíveis

➤ SiFive (<https://sifive.com>):

- Fundadores fazem parte da equipe criadora da arquitetura
- System On Chip SiFive Freedom (capazes de rodar Linux):
 - HiFive Unmatched: 16GB DDR4, MicroSD e M.2, com slot PCIe
 - HiFive Unleashed: 8GB DDR4, MicroSD (descontinuado)
- Micro Controlador: HiFive1

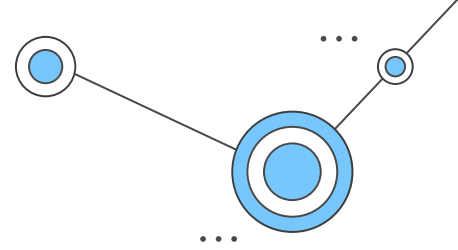
➤ BeagleBoard (<https://beagleboard.org>):

- BeagleV

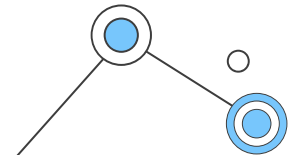
➤ Outras empresas: <https://riscv.org/exchange/boards/>



Software Disponível



- Bastante coisa crucial já funciona:
 - Emulação: QEMU e muitos outros...
 - Toolchain: LLVM, Binutils...
 - Compiladores e Libs C: LLVM, GCC, Clang, Glibc, Musl...
 - Bootloaders: U-Boot, Coreboot...
 - Outros compiladores e Runtimes: Go, OCaml, OpenJDK, Rust, NodeJS, Mono, Ada, Free Pascal...
 - E muito mais: <https://riscv.org/exchange/software/>



Linux e BSDs

Algumas distros já podem executar bem em hardware
(normalmente da SiFive) ou em VMs:

- Debian: <https://wiki.debian.org/RISC-V>
- Ubuntu: <https://wiki.ubuntu.com/RISC-V>
- Gentoo: <https://wiki.gentoo.org/wiki/Project:RISC-V>
- Fedora: <http://fedoraproject.org/wiki/Architectures/RISC-V>
- NixOS: <https://github.com/NixOS/nixpkgs/issues/101651>
- Parabola: <https://github.com/oaken-source/parabola-cross-bootstrap>
- FreeBSD: <https://wiki.freebsd.org/riscv>

Obrigada!

Referências:

- <https://riscv.org/wp-content/uploads/2017/05/riscv-spec-v2.2.pdf>
- <https://riscv.org/about/history/>
- <https://riscv.org/about/>
- <https://en.wikipedia.org/wiki/RISC-V>
- <https://riscv.org/exchange/boards/>
- <https://riscv.org/exchange/software/>

Repositório:

- <https://github.com/Misterio77/BSI-SSC0510>

