ORGANIZAÇÃO E ARQUITETURA DE COMPUTADORES

Arquitetura RISC-V

Profª. Fabiana F F Peres

Apoio: Camile Bordini

Conceitos: Arquitetura

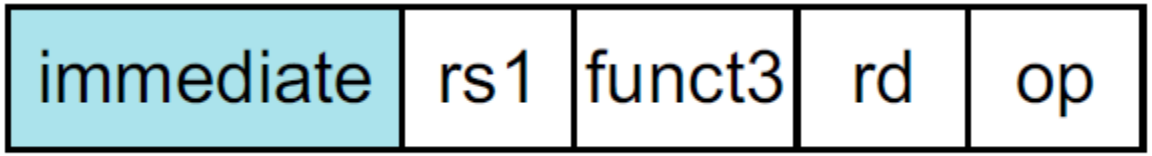
"Conjunto de recursos percebidos pelo programador em linguagem de máquina, tais como

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

Modos de Endereçamento RISC-V

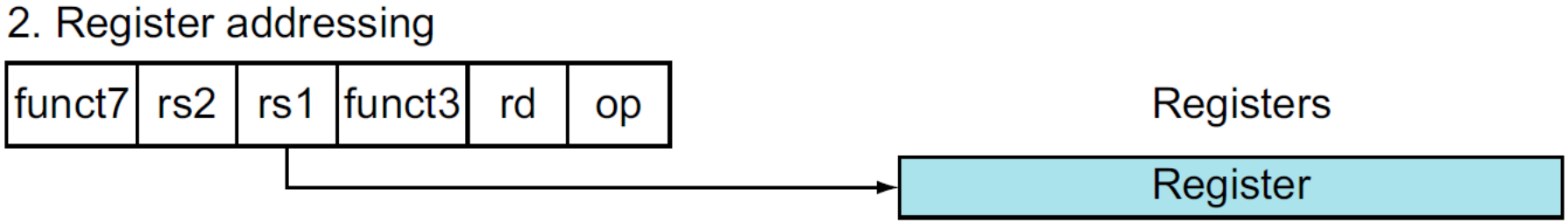
- **Endereçamento imediato**:
  - Operando da instrução é uma **constante**
  - A instrução contém o operando especificado num campo da instrução
- Ex.: **`addi x10, x21,`** **`100`** `//x10=x21+100`

1. Immediate addressing



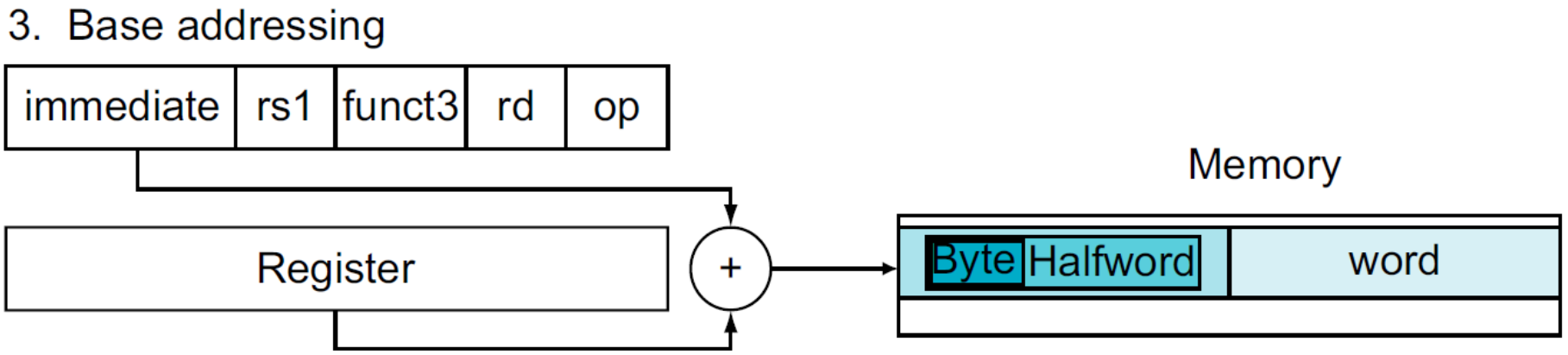
Modos de Endereçamento RISC-V

- **Endereçamento de registrador:**
  - O operando é um **registrador**;
  - Um campo contém o endereço (o número) do registrador no qual o operando está armazenado
- Ex: **`add x10, x21, x25`**



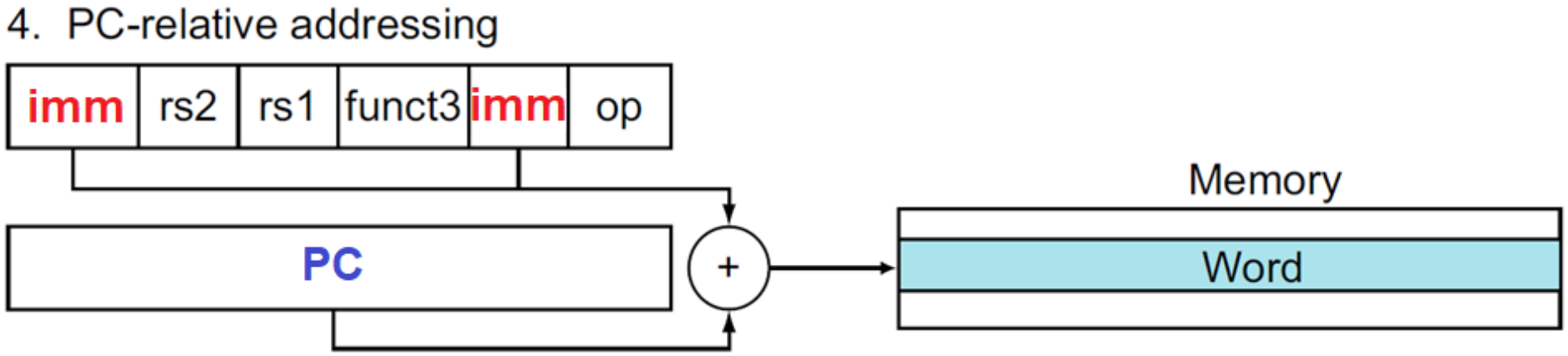
Modos de Endereçamento RISC-V

- **Endereçamento de base:**
  - O operando está numa posição da memória onde o endereço é a **soma** do conteúdo de um **registrador de base** com uma **constante da instrução**
- Ex: `lw x21, 100(x25)`



Modos de Endereçamento RISC-V

- **Endereçamento relativo a PC:**
  - O endereço é a soma do PC e uma constante da instrução
- Ex.: **`beq x20, x21, 12`**



Questão

What RISC-V instruction does this represent? Choose from one of the four options below.

funct7	rs2	rs1	funct3	rd	opcode
32	9	10	000	11	51

1. `sub x9, x10, x11`
2. `add x11, x9, x10`
3. `sub x11, x10, x9`
4. `sub x11, x9, x10`

Questão

What RISC-V instruction does this represent? Choose from one of the four options below.

funct7	rs2	rs1	funct3	rd	opcode
32	9	10	000	11	51

1. `sub x9, x10, x11`
2. `add x11, x9, x10`
3. `sub x11, x10, x9`
4. `sub x11, x9, x10`

Conjunto de Instruções

Classe de instruções	Instruções	Tipo
Instruções aritméticas	`add rd, rs1, rs2`	R
	`sub rd, rs1, rs2`	R
	`addi rd, rs1, imm`	I
Instruções de transferência de dados	`lw rd, imm(rs1)`	I
	`ld rd, imm(rs1)`	I
	`sw rs2, imm(rs1)`	S
	`sd rs2, imm(rs1)`	S
Instruções lógicas	`or rd, rs1, rs2`	R
	`and rd, rs1, rs2`	R
	`xor rd, rs1, rs2`	R
Instruções de deslocamentos	`sll rd, rs1, rs2`	R
	`srl rd, rs1, rs2`	R
	`slli rd, rs1, imm`	I
Instruções de desvios condicionais	`bne rs1, rs2, L`	B
	`beq rs1, rs2, L`	B
Instruções de desvios incondicionais	`jal rd, imm`	J
	`jalr rd, imm(rs1)`	I

RISC-V Instructions	Name	Format
Add	add	R
Subtract	sub	R
Add immediate	addi	I
Load doubleword	ld	I
Store doubleword	sd	S
Load word	lw	I
Load word, unsigned	lwu	I
Store word	sw	S
Load halfword	lh	I
Load halfword, unsigned	lhu	I
Store halfword	sh	S
Load byte	lb	I
Load byte, unsigned	lbu	I
Store byte	sb	S
Load reserved	lr.d	R
Store conditional	sc.d	R
Load upper immediate	lui	U

RISC-V Instructions	Name	Format
And	and	R
Inclusive or	or	R
Exclusive or	xor	R
And immediate	andi	I
Inclusive or immediate	ori	I
Exclusive or immediate	xori	I
Shift left logical	sll	R
Shift right logical	srl	R
Shift right arithmetic	sra	R
Shift left logical immediate	slli	I
Shift right logical immediate	srli	I
Shift right arithmetic immediate	srai	I
Branch if equal	beq	SB
Branch if not equal	bne	SB
Branch if less than	blt	SB
Branch if greater or equal	bge	SB
Branch if less, unsigned	bltu	SB
Branch if greatr/eq, unsigned	bgeu	SB
Jump and link	jal	UJ
Jump and link register	jalr	I

Conjunto de Instruções

1. Instruções aritméticas
2. Instruções de transferência de dados
3. Instruções lógicas
4. Instruções de deslocamentos
5. **Instruções de desvios condicionais**
6. Instruções de desvios incondicionais

Instruções de Desvios

- Decisões em linguagens de programação:
  - `if`
  - `go-to`

- A linguagem assembly RISC-V inclui duas instruções de tomada de decisão semelhantes:
  - ***`beq`****: branch if equal*
  - ***`bne`****: branch if not equal*

Instruções de Desvios

- **`beq rs1, rs2, L1`**

//Vá para a instrução L1 se o valor no registrador rs1 for igual ao do rs2

- **`bne rs1, rs2, L1`**

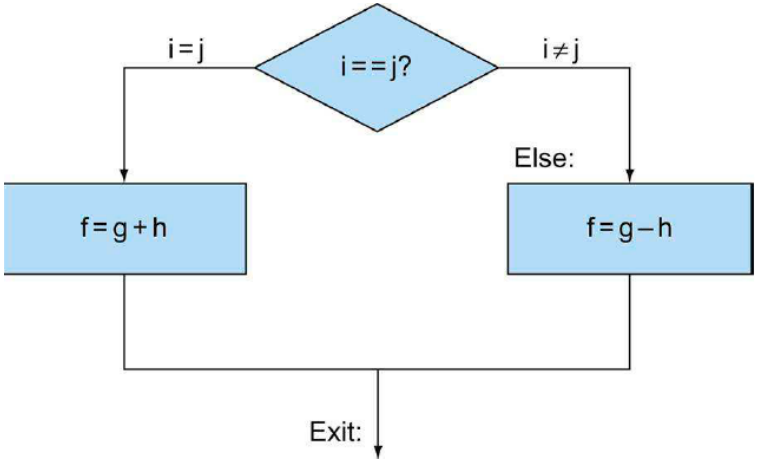
//Vá para a instrução L1 se o valor no registrador rs1 ***não*** *for igual ao do rs2*

Instruction	Example	Meaning
Branch if equal	`beq x5, x6, 100`	`if (x5 == x6) go to PC+100`
Branch if not equal	`bne x5, x6, 100`	`if (x5 != x6) go to PC+100`

Exemplo if

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

```
if (i == j)  f = g + h;
else f = g - h;
```

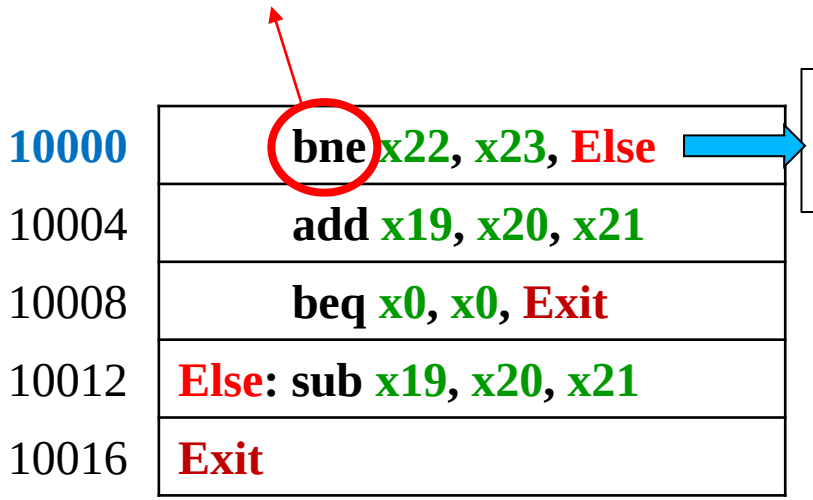


- Qual o Código em RISC-V correspondente?

```
add rd, rs1, rs2
sub rd, rs1, rs2
lw rd, imm(rs1)
sw rs2, imm(rs1)
addi rd, rs1, imm
bne rs1, rs2, L
beq rs1, rs2, L
```

```
if (i == j)  f = g + h;
else f = g - h;
```

- Em geral, o código é mais eficiente se testarmos a condição oposta, que ramifica se os valores não forem iguais



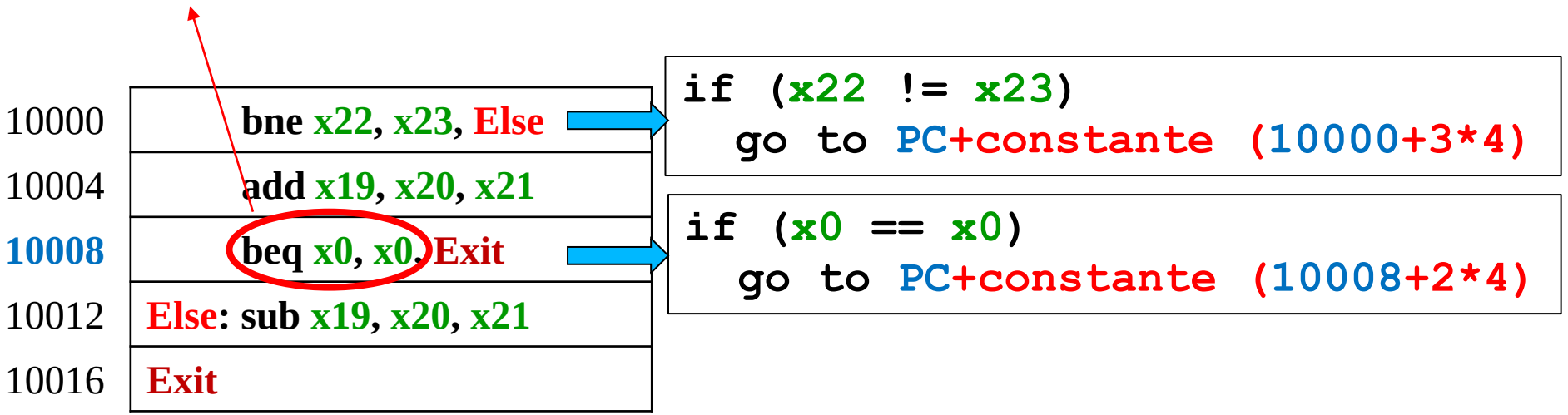

```
if (x22 != x23)
  go to PC+constante (10000+3*4)
```

```
constante = numero de
instruções saltadas*4
```

***Endereçamento Relativo a PC*:** o salto é realizado com base no endereço atual do registrador PC – *Program Counter*

```
if (i == j)  f = g + h;
else f = g - h;
```

- **Desvio condicional** que é sempre verdade



- Diferente de **load's** e **store's**, em **desvios** o montador livra o compilador e o programador assembly de calcular endereços para as ramificações, através do uso de *label's*

Exemplo if

```
if (i == j)   f = g + h;
else f = g - h;
```

```
        (Tipo-B)bne x22, x23, Else
        (Tipo-R)add x19, x20, x21
        (Tipo-B)beq x0, x0, Exit
Else:   (Tipo-R)sub x19, x20, x21
Exit:
```

Tipo-B	*constante*	*23*	*22*	*funct3*	*constante*	*opcode*
Tipo-R	*funct7*	***21***	***20***	*funct3*	***19***	*opcode*
Tipo-B	*constante*	*0*	*0*	*funct3*	*constante*	*opcode*
Tipo-R	*funct7*	***21***	***20***	*funct3*	***19***	*opcode*

Instruções de Desvios

- Decisões são importantes não só para escolher entre duas alternativas (*if's*) quanto para **iterar** uma computação (*loop's*)

- Exemplo: considere que **`i`** e **`k`** correspondem aos registradores **x22** e **x24**, e a base/segmento do array **`save`** está em **x25**. Qual o código em assembly RISC-V?

```
while (save[i] == k)
   i += 1;
```

Exemplo while

```
while (save[i] == k)
   i += 1;
```

```
Loop: slli x10, x22, 2      //temp x10 = i * 4
      add x10, x10, x25     //x10 = endereço de save[i]
      lw x9, 0(x10)         //temp x9 = save[i]
      bne x9, x24, Exit     //vá para Exit se save[i]!=k
      addi x22, x22, 1      //i = i + 1
      beq x0, x0, Loop      //vá para Loop
Exit:
```

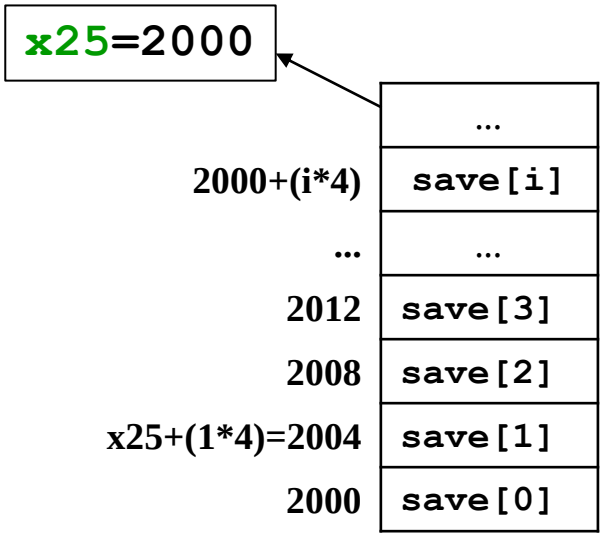
Exemplo while

Instruções	Tipo
`add rd, rs1, rs2`	R
`sub rd, rs1, rs2`	R
`addi rd, rs1, imm`	I
`lw rd, imm(rs1)`	I
`sw rs2, imm(rs1)`	S
`or rd, rs1, rs2`	R
`and rd, rs1, rs2`	R
`xor rd, rs1, rs2`	R

Instruções	Tipo
`sll rd, rs1, rs2`	R
`srl rd, rs1, rs2`	R
`slli rd, rs1, imm`	I
`bne rs1, rs2, L`	B
`beq rs1, rs2, L`	B
`jal rd, imm`	J
`jalr rd, imm(rs1)`	I

Tipo-R	*funct7*	***rs2***	***rs1***	*funct3*	***rd***	*opcode*
Tipo-I	***imm[11:0]***		***rs1***	*funct3*	***rd***	*opcode*
Tipo-S	***imm[11:5]***	***rs2***	***rs1***	*funct3*	***imm[4:0]***	*opcode*
Tipo-B	***imm[12] | imm[10:5]***	***rs2***	***rs1***	*funct3*	***imm[4:1] | imm[11]***	*opcode*
Tipo-U	***imm[31-12]***				***rd***	*opcode*
Tipo-J	***imm[20] | imm[10:1] | imm[11] | imm[19:12]***				***rd***	*opcode*

Exemplo while




```
Loop: slli x10, x22, 2          (I)
      add x10, x10, x25         (R)
      lw x9, 0(x10)             (I)
      bne x9, x24, Exit (+3)    (B)
      addi x22, x22, 1          (I)
      beq x0, x0, Loop (-5)     (B)
Exit:
```

Tipo-R	*funct7*	*rs2*	*rs1*	*funct3*	*rd*	*opcode*
Tipo-I	*imm[11:0]*		*rs1*	*funct3*	*rd*	*opcode*
Tipo-S	*imm[11:5]*	*rs2*	*rs1*	*funct3*	*imm[4:0]*	*opcode*
Tipo-B	*imm[12] | imm[10:5]*	*rs2*	*rs1*	*funct3*	*imm[4:1] | imm[11]*	*opcode*
Tipo-U	*imm[31-12]*				*rd*	*opcode*
Tipo-J	*imm[20] | imm[10:1] | imm[11] | imm[19:12]*				*rd*	*opcode*

- Assumindo que o início do código do loop está no endereço 80.000 na memória

80000	Loop: slli x10, x22, 2
80004	add x10, x10, x25
80008	lw x9, 0(x25)
80012	bne x9, x24, Exit //80012 + (12)
80016	addi x22, x22, 1
80020	beq x0, x0, Loop //80020 + (-20)
80024	Exit:

Tipo-I	LOOP:slli,x10,x22,2	2		22	1	10	19
Tipo-R	add x10,x10,x25	0	25	10	0	10	51
Tipo-I	lw x9,0(x10)	0		10	2	9	3
Tipo-B	bne x9, x24,EXIT	0	24	9	1	12	99
Tipo-I	addi x22, x22, 1	1		22	0	22	19
Tipo-B	beq x0,x0,LOOP	127	0	0	0	13	99

Exemplo while

Endereço	Instrução					
80000	2		22	1	10	19
80004	0	25	10	0	10	51
80008	0		10	2	9	3
80012	0	24	9	1	12	99
80016	1		22	0	22	19
80020	127	0	0	0	13	99

Endereço	Instrução					
80000	000000000010		10110	001	01010	0010011
80004	0000000	11001	01010	000	01010	0110011
80008	000000000000		01010	011	01001	0000011
80012	0000000	11000	01001	001	01100	1100011
80016	000000000001		10110	000	10110	0010011
80020	1111111	00000	00000	000	01101	1100011

Tipo-R	*funct7*	*rs2*	*rs1*	*funct3*	*rd*	*opcode*
Tipo-I	*imm[11:0]*		*rs1*	*funct3*	*rd*	*opcode*
Tipo-S	*imm[11:5]*	*rs2*	*rs1*	*funct3*	*imm[4:0]*	*opcode*
Tipo-B	*imm[12]|imm[10:5]*	*rs2*	*rs1*	*funct3*	*imm[4:1]|imm[11]*	*opcode*
Tipo-U	*imm[31-12]*				*rd*	*opcode*
Tipo-J	*imm[20]|imm[10:1]|imm[11]|imm[19:12]*				*rd*	*opcode*

Tipo-R para **operações** de registradores

Instruções
`add rd, rs1, rs2`
`sub rd, rs1, rs2`
`addi rd, rs1, imm`
`lw rd, imm(rs1)`
`ld rd, imm(rs1)`
`sw rs2, imm(rs1)`
`sd rs2, imm(rs1)`
`or rd, rs1, rs2`
`and rd, rs1, rs2`

Instruções
`xor rd, rs1, rs2`
`sll rd, rs1, rs2`
`srl rd, rs1, rs2`
`slli rd,rs1, imm`
`bne rs1, rs2, L`
`beq rs1, rs2, L`
`jal rd, imm`
`jalr rd, imm(rs1)`

Tipo-R	*funct7*	*rs2*	*rs1*	*funct3*	*rd*	*opcode*
Tipo-I	***imm[11:0]***		***rs1***	***funct3***	***rd***	***opcode***
Tipo-S	*imm[11:5]*	*rs2*	*rs1*	*funct3*	*imm[4:0]*	*opcode*
Tipo-B	*imm[12]|imm[10:5]*	*rs2*	*rs1*	*funct3*	*imm[4:1]|imm[11]*	*opcode*
Tipo-U	*imm[31-12]*				*rd*	*opcode*
Tipo-J	*imm[20]|imm[10:1]|imm[11]|imm[19:12]*				*rd*	*opcode*

Tipo-I para **valores imediatos curtos** e **loads**

Instruções
`add rd, rs1, rs2`
`sub rd, rs1, rs2`
`addi rd, rs1, imm`
`lw rd, imm(rs1)`
`ld rd, imm(rs1)`
`sw rs2, imm(rs1)`
`sd rs2, imm(rs1)`
`or rd, rs1, rs2`
`and rd, rs1, rs2`

Instruções
`xor rd, rs1, rs2`
`sll rd, rs1, rs2`
`srl rd, rs1, rs2`
`slli rd,rs1, imm`
`bne rs1, rs2, L`
`beq rs1, rs2, L`
`jal rd, imm`
`jalr rd, imm(rs1)`

Tipo-R	*funct7*	*rs2*	*rs1*	*funct3*	*rd*	*opcode*
Tipo-I	*imm[11:0]*		*rs1*	*funct3*	*rd*	*opcode*
Tipo-S	***imm[11:5]***	***rs2***	***rs1***	***funct3***	***imm[4:0]***	***opcode***
Tipo-B	*imm[12]|imm[10:5]*	*rs2*	*rs1*	*funct3*	*imm[4:1]|imm[11]*	*opcode*
Tipo-U	*imm[31-12]*				*rd*	*opcode*
Tipo-J	*imm[20]|imm[10:1]|imm[11]|imm[19:12]*				*rd*	*opcode*

Tipo-S para **stores**

Instruções
`add rd, rs1, rs2`
`sub rd, rs1, rs2`
`addi rd, rs1, imm`
`lw rd, imm(rs1)`
`ld rd, imm(rs1)`
`sw rs2, imm(rs1)`
`sd rs2, imm(rs1)`
`or rd, rs1, rs2`
`and rd, rs1, rs2`

Instruções
`xor rd, rs1, rs2`
`sll rd, rs1, rs2`
`srl rd, rs1, rs2`
`slli rd,rs1, imm`
`bne rs1, rs2, L`
`beq rs1, rs2, L`
`jal rd, imm`
`jalr rd, imm(rs1)`

Tipo-R	*funct7*	*rs2*	*rs1*	*funct3*	*rd*	*opcode*
Tipo-I	*imm[11:0]*		*rs1*	*funct3*	*rd*	*opcode*
Tipo-S	*imm[11:5]*	*rs2*	*rs1*	*funct3*	*imm[4:0]*	*opcode*
Tipo-B	***imm[12]|imm[10:5]***	***rs2***	***rs1***	***funct3***	***imm[4:1]|imm[11]***	***opcode***
Tipo-U	*imm[31-12]*				*rd*	*opcode*
Tipo-J	*imm[20]|imm[10:1]|imm[11]|imm[19:12]*				*rd*	*opcode*

Tipo-B para **desvios condicionais**

Instruções
`add rd, rs1, rs2`
`sub rd, rs1, rs2`
`addi rd, rs1, imm`
`lw rd, imm(rs1)`
`ld rd, imm(rs1)`
`sw rs2, imm(rs1)`
`sd rs2, imm(rs1)`
`or rd, rs1, rs2`
`and rd, rs1, rs2`

Instruções
`xor rd, rs1, rs2`
`sll rd, rs1, rs2`
`srl rd, rs1, rs2`
`slli rd,rs1, imm`
`bne rs1, rs2, L`
`beq rs1, rs2, L`
`jal rd, imm`
`jalr rd, imm(rs1)`

Tipo-R	*Funct7*	*rs2*	*rs1*	*funct3*	*rd*	*opcode*
Tipo-I	*imm[11:0]*		*rs1*	*funct3*	*rd*	*opcode*
Tipo-S	*imm[11:5]*	*rs2*	*rs1*	*funct3*	*imm[4:0]*	*opcode*
Tipo-B	*imm[12]|imm[10:5]*	*rs2*	*rs1*	*funct3*	*imm[4:1]|imm[11]*	*opcode*
Tipo-U	***imm[31-12]***				***rd***	***opcode***
Tipo-J	***imm[20]|imm[10:1]|imm[11]|imm[19:12]***				***rd***	***opcode***

Tipo-J para **saltos incondicionais**

Instruções
`add rd, rs1, rs2`
`sub rd, rs1, rs2`
`addi rd, rs1, imm`
`lw rd, imm(rs1)`
`ld rd, imm(rs1)`
`sw rs2, imm(rs1)`
`sd rs2, imm(rs1)`
`or rd, rs1, rs2`
`and rd, rs1, rs2`

Instruções
`xor rd, rs1, rs2`
`sll rd, rs1, rs2`
`srl rd, rs1, rs2`
`slli rd,rs1, imm`
`bne rs1, rs2, L`
`beq rs1, rs2, L`
`jal rd, imm`
`jalr rd, imm(rs1)`

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.