ORGANIZAÇÃO E ARQUITETURA DE COMPUTADORES

Arquitetura RISC-V

Profª. Fabiana F F Peres

Apoio: Camile Bordini

Conceitos: Arquitetura

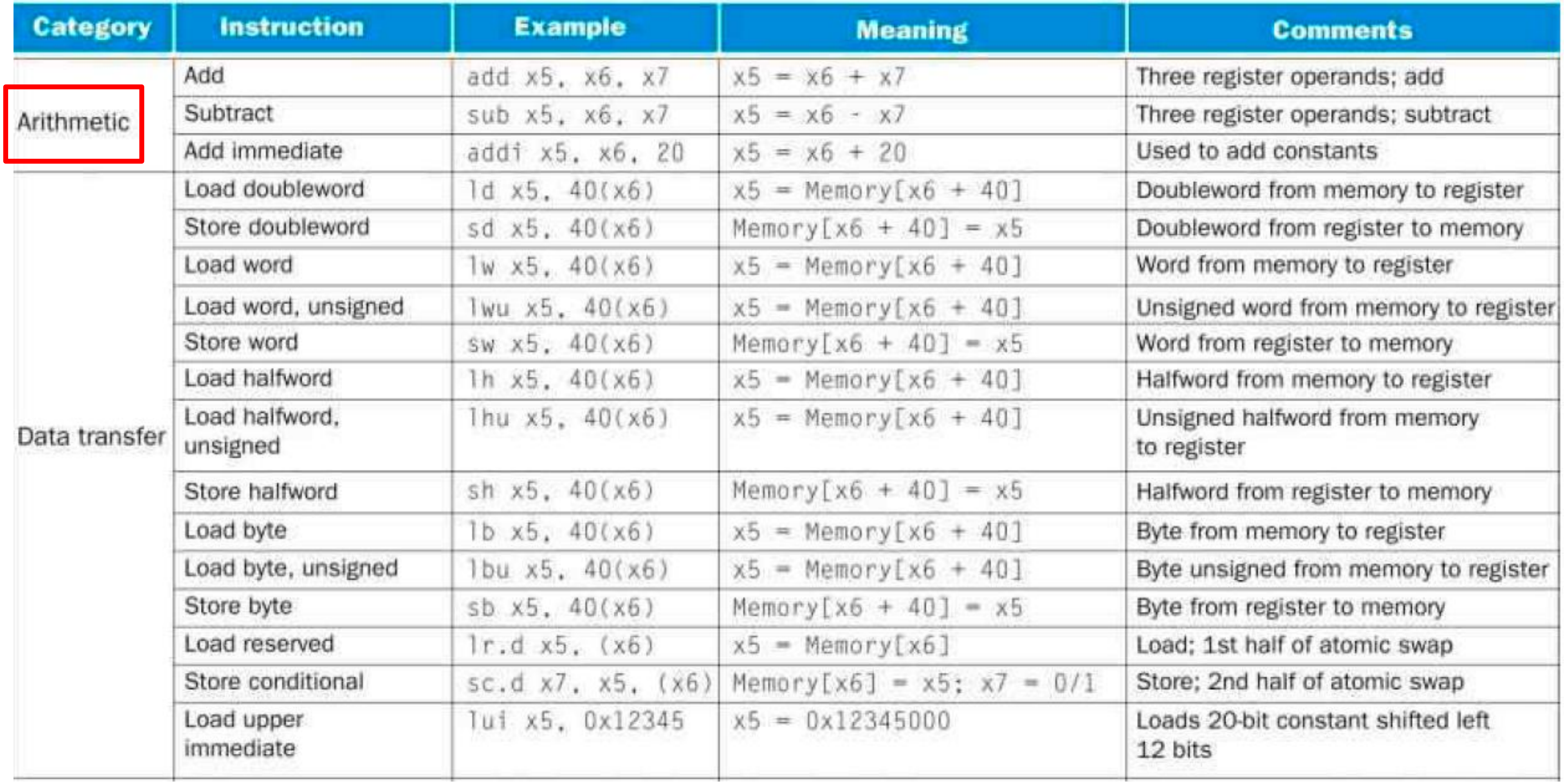
“Conjunto de recursos percebidos pelo programador em linguagem de máquina, tais como

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

Conjunto de Instruções

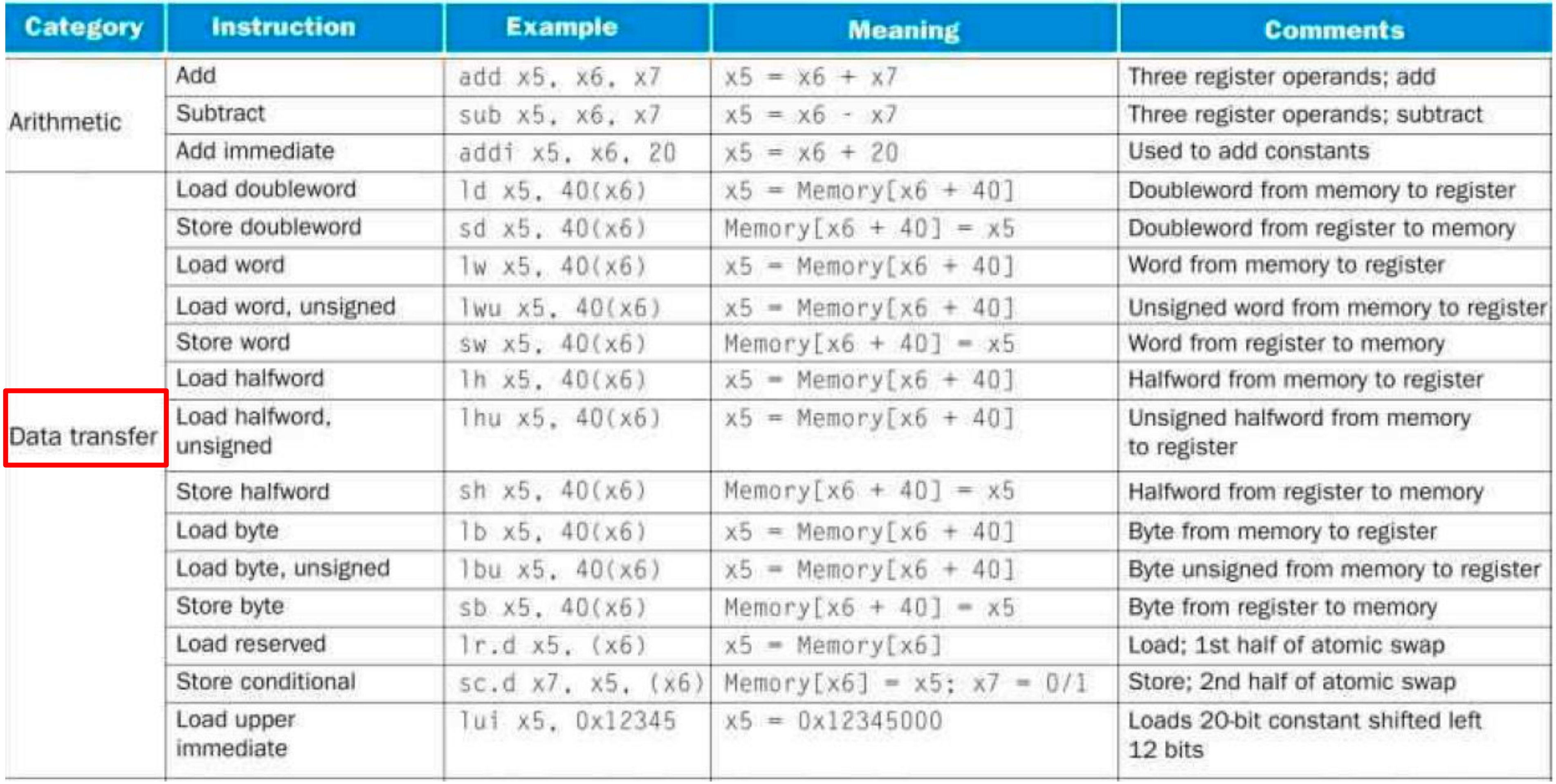
1. **Instruções aritméticas**
2. **Instruções de transferência de dados**
3. **Instruções lógicas**
4. **Instruções de deslocamentos**
5. **Instruções de desvios condicionais**
6. **Instruções de desvios incondicionais**

1. Instruções aritméticas



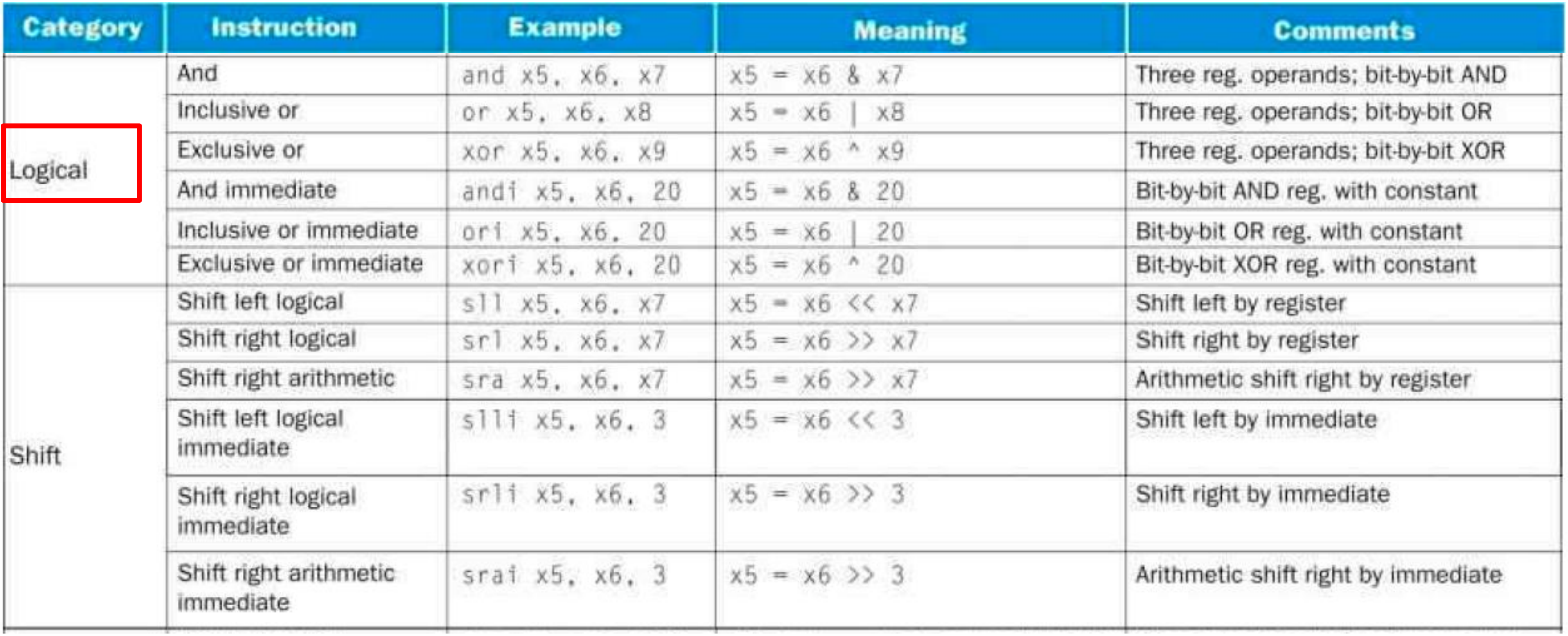
Category	Instruction	Example	Meaning	Comments
Arithmetic	Add	`add x5, x6, x7`	`x5 = x6 + x7`	Three register operands; add
	Subtract	`sub x5, x6, x7`	`x5 = x6 - x7`	Three register operands; subtract
	Add immediate	`addi x5, x6, 20`	`x5 = x6 + 20`	Used to add constants
Data transfer	Load doubleword	`ld x5, 40(x6)`	`x5 = Memory[x6 + 40]`	Doubleword from memory to register
	Store doubleword	`sd x5, 40(x6)`	`Memory[x6 + 40] = x5`	Doubleword from register to memory
	Load word	`lw x5, 40(x6)`	`x5 = Memory[x6 + 40]`	Word from memory to register
	Load word, unsigned	`lwu x5, 40(x6)`	`x5 = Memory[x6 + 40]`	Unsigned word from memory to register
	Store word	`sw x5, 40(x6)`	`Memory[x6 + 40] = x5`	Word from register to memory
	Load halfword	`lh x5, 40(x6)`	`x5 = Memory[x6 + 40]`	Halfword from memory to register
	Load halfword, unsigned	`lhu x5, 40(x6)`	`x5 = Memory[x6 + 40]`	Unsigned halfword from memory to register
	Store halfword	`sh x5, 40(x6)`	`Memory[x6 + 40] = x5`	Halfword from register to memory
	Load byte	`lb x5, 40(x6)`	`x5 = Memory[x6 + 40]`	Byte from memory to register
	Load byte, unsigned	`lbu x5, 40(x6)`	`x5 = Memory[x6 + 40]`	Byte unsigned from memory to register
	Store byte	`sb x5, 40(x6)`	`Memory[x6 + 40] = x5`	Byte from register to memory
	Load reserved	`lr.d x5, (x6)`	`x5 = Memory[x6]`	Load; 1st half of atomic swap
	Store conditional	`sc.d x7, x5, (x6)`	`Memory[x6] = x5; x7 = 0/1`	Store; 2nd half of atomic swap
	Load upper immediate	`lui x5, 0x12345`	`x5 = 0x12345000`	Loads 20-bit constant shifted left 12 bits

2. Instruções de transferência de dados



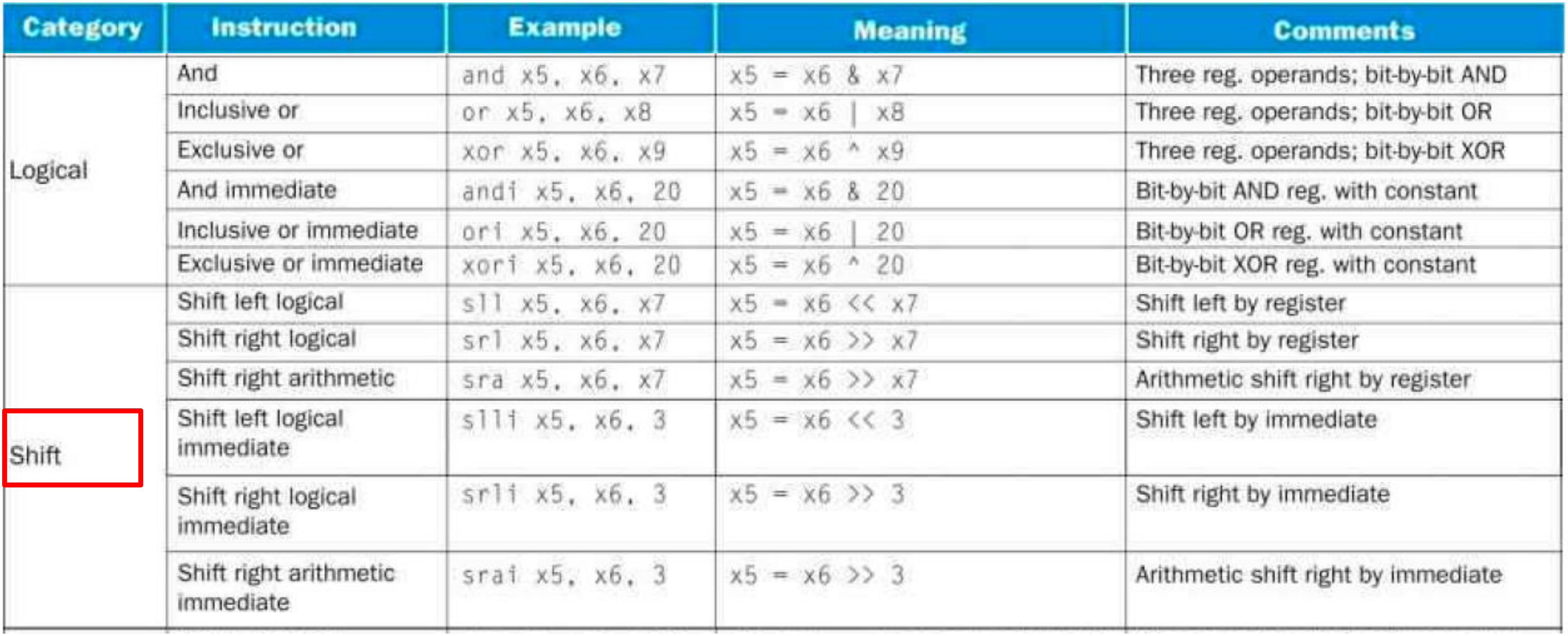
Category	Instruction	Example	Meaning	Comments
Arithmetic	Add	add x5, x6, x7	x5 = x6 + x7	Three register operands; add
	Subtract	sub x5, x6, x7	x5 = x6 - x7	Three register operands; subtract
	Add immediate	addi x5, x6, 20	x5 = x6 + 20	Used to add constants
Data transfer	Load doubleword	ld x5, 40(x6)	x5 = Memory[x6 + 40]	Doubleword from memory to register
	Store doubleword	sd x5, 40(x6)	Memory[x6 + 40] = x5	Doubleword from register to memory
	Load word	lw x5, 40(x6)	x5 = Memory[x6 + 40]	Word from memory to register
	Load word, unsigned	lwu x5, 40(x6)	x5 = Memory[x6 + 40]	Unsigned word from memory to register
	Store word	sw x5, 40(x6)	Memory[x6 + 40] = x5	Word from register to memory
	Load halfword	lh x5, 40(x6)	x5 = Memory[x6 + 40]	Halfword from memory to register
	Load halfword, unsigned	lhu x5, 40(x6)	x5 = Memory[x6 + 40]	Unsigned halfword from memory to register
	Store halfword	sh x5, 40(x6)	Memory[x6 + 40] = x5	Halfword from register to memory
	Load byte	lb x5, 40(x6)	x5 = Memory[x6 + 40]	Byte from memory to register
	Load byte, unsigned	lbu x5, 40(x6)	x5 = Memory[x6 + 40]	Byte unsigned from memory to register
	Store byte	sb x5, 40(x6)	Memory[x6 + 40] = x5	Byte from register to memory
	Load reserved	lr.d x5, (x6)	x5 = Memory[x6]	Load; 1st half of atomic swap
	Store conditional	sc.d x7, x5, (x6)	Memory[x6] = x5; x7 = 0/1	Store; 2nd half of atomic swap
	Load upper immediate	lui x5, 0x12345	x5 = 0x12345000	Loads 20-bit constant shifted left 12 bits

3. Instruções lógicas



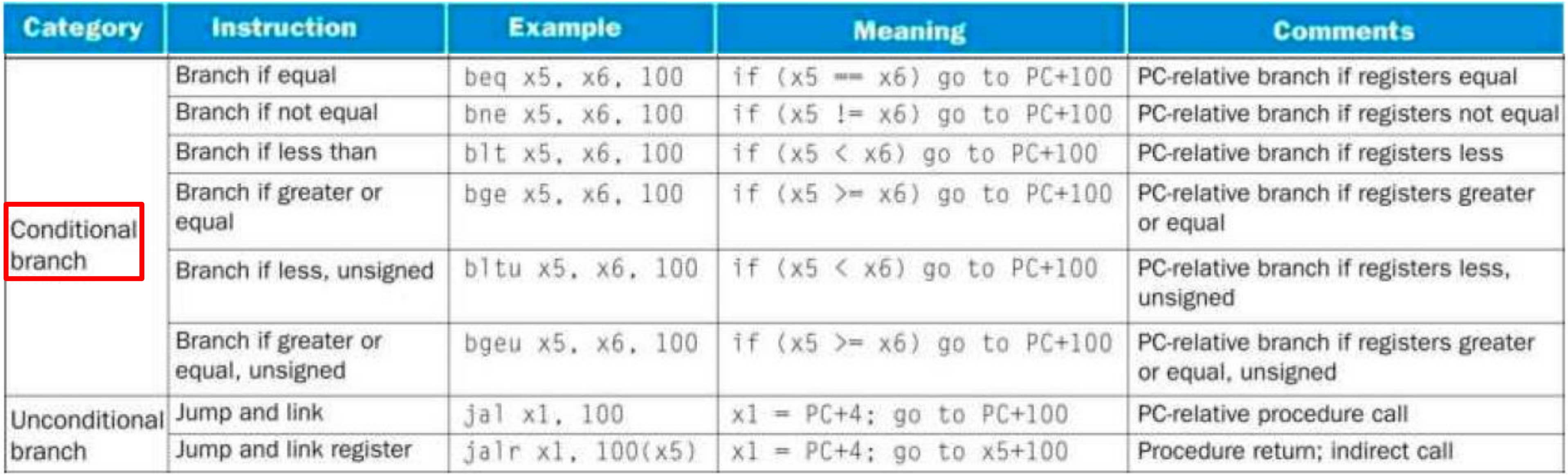
Category	Instruction	Example	Meaning	Comments
Logical	And	and x5, x6, x7	x5 = x6 & x7	Three reg. operands; bit-by-bit AND
	Inclusive or	or x5, x6, x8	x5 = x6 | x8	Three reg. operands; bit-by-bit OR
	Exclusive or	xor x5, x6, x9	x5 = x6 ^ x9	Three reg. operands; bit-by-bit XOR
	And immediate	andi x5, x6, 20	x5 = x6 & 20	Bit-by-bit AND reg. with constant
	Inclusive or immediate	ori x5, x6, 20	x5 = x6 | 20	Bit-by-bit OR reg. with constant
	Exclusive or immediate	xori x5, x6, 20	x5 = x6 ^ 20	Bit-by-bit XOR reg. with constant
Shift	Shift left logical	sll x5, x6, x7	x5 = x6 << x7	Shift left by register
	Shift right logical	srl x5, x6, x7	x5 = x6 >> x7	Shift right by register
	Shift right arithmetic	sra x5, x6, x7	x5 = x6 >> x7	Arithmetic shift right by register
	Shift left logical immediate	slli x5, x6, 3	x5 = x6 << 3	Shift left by immediate
	Shift right logical immediate	srli x5, x6, 3	x5 = x6 >> 3	Shift right by immediate
	Shift right arithmetic immediate	srai x5, x6, 3	x5 = x6 >> 3	Arithmetic shift right by immediate

4. Instruções de deslocamentos



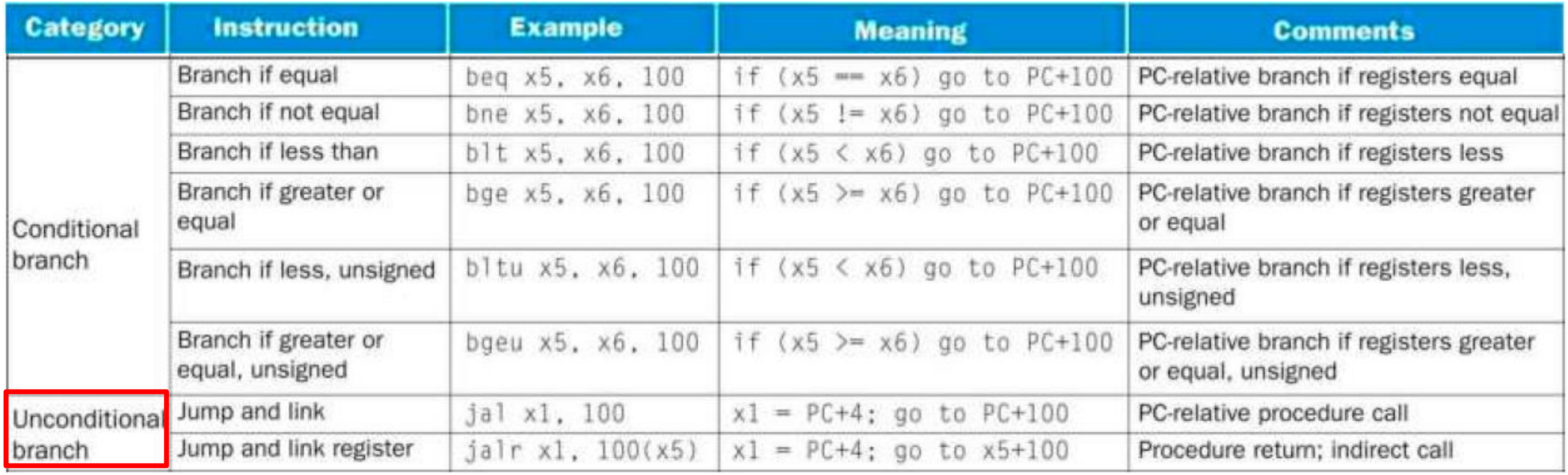
Category	Instruction	Example	Meaning	Comments
Logical	And	and x5, x6, x7	x5 = x6 & x7	Three reg. operands; bit-by-bit AND
	Inclusive or	or x5, x6, x8	x5 = x6 | x8	Three reg. operands; bit-by-bit OR
	Exclusive or	xor x5, x6, x9	x5 = x6 ^ x9	Three reg. operands; bit-by-bit XOR
	And immediate	andi x5, x6, 20	x5 = x6 & 20	Bit-by-bit AND reg. with constant
	Inclusive or immediate	ori x5, x6, 20	x5 = x6 | 20	Bit-by-bit OR reg. with constant
	Exclusive or immediate	xori x5, x6, 20	x5 = x6 ^ 20	Bit-by-bit XOR reg. with constant
Shift	Shift left logical	sll x5, x6, x7	x5 = x6 << x7	Shift left by register
	Shift right logical	srl x5, x6, x7	x5 = x6 >> x7	Shift right by register
	Shift right arithmetic	sra x5, x6, x7	x5 = x6 >> x7	Arithmetic shift right by register
	Shift left logical immediate	slli x5, x6, 3	x5 = x6 << 3	Shift left by immediate
	Shift right logical immediate	srli x5, x6, 3	x5 = x6 >> 3	Shift right by immediate
	Shift right arithmetic immediate	srai x5, x6, 3	x5 = x6 >> 3	Arithmetic shift right by immediate

5. Instruções de desvios condicionais



Category	Instruction	Example	Meaning	Comments
Conditional branch	Branch if equal	`beq x5, x6, 100`	`if (x5 == x6) go to PC+100`	PC-relative branch if registers equal
	Branch if not equal	`bne x5, x6, 100`	`if (x5 != x6) go to PC+100`	PC-relative branch if registers not equal
	Branch if less than	`blt x5, x6, 100`	`if (x5 < x6) go to PC+100`	PC-relative branch if registers less
	Branch if greater or equal	`bge x5, x6, 100`	`if (x5 >= x6) go to PC+100`	PC-relative branch if registers greater or equal
	Branch if less, unsigned	`bltu x5, x6, 100`	`if (x5 < x6) go to PC+100`	PC-relative branch if registers less, unsigned
	Branch if greater or equal, unsigned	`bgeu x5, x6, 100`	`if (x5 >= x6) go to PC+100`	PC-relative branch if registers greater or equal, unsigned
Unconditional branch	Jump and link	`jal x1, 100`	`x1 = PC+4; go to PC+100`	PC-relative procedure call
	Jump and link register	`jalr x1, 100(x5)`	`x1 = PC+4; go to x5+100`	Procedure return; indirect call

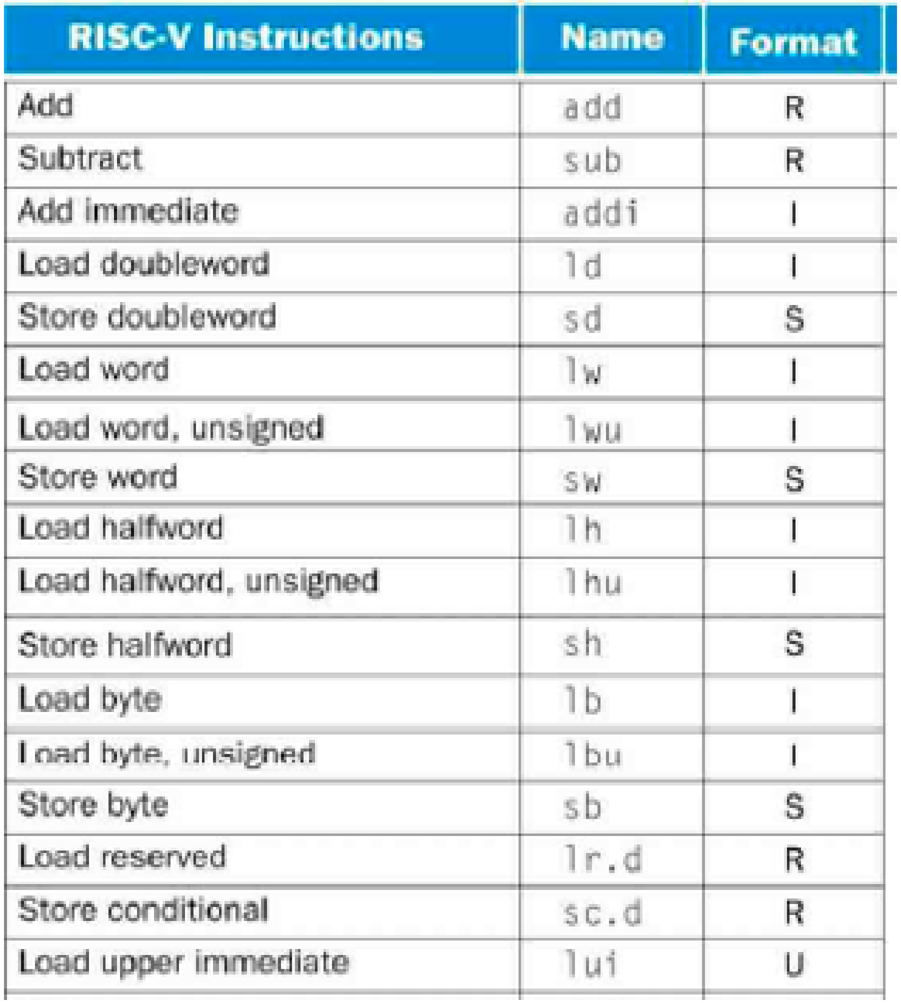
6. Instruções de desvios incondicionais



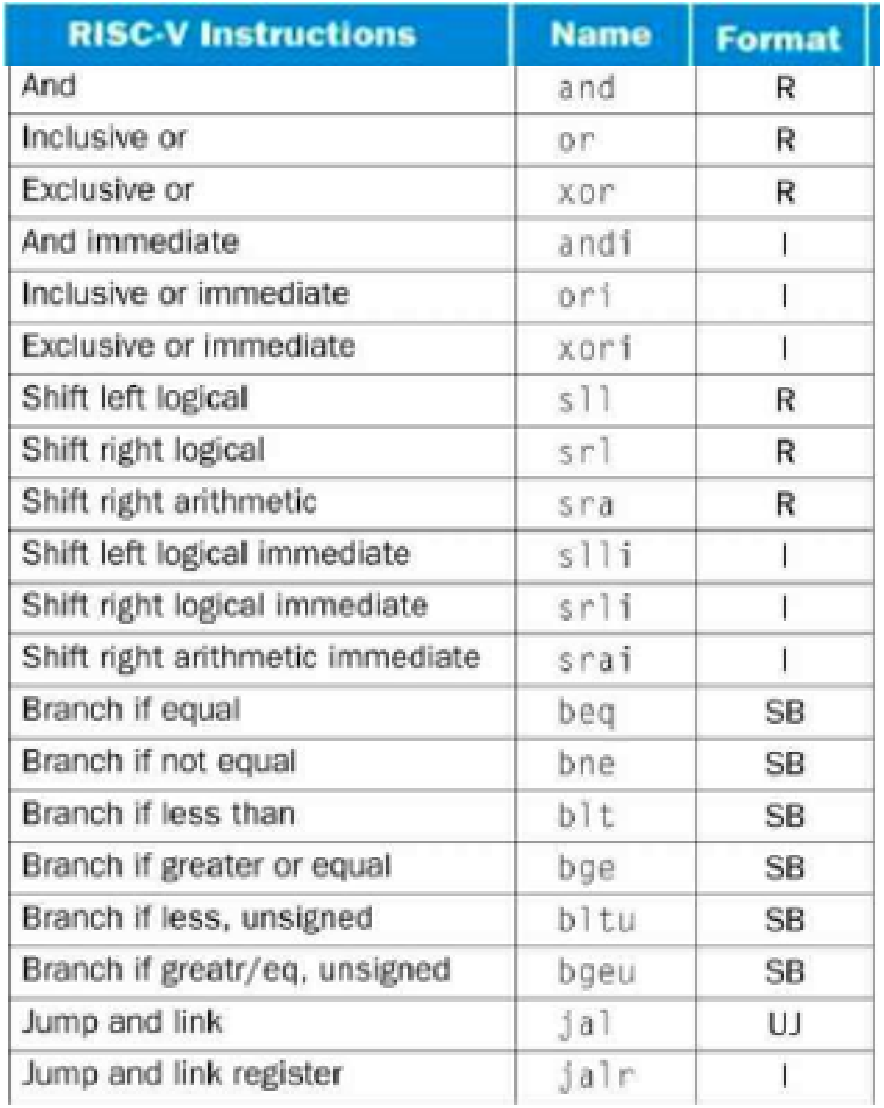
Category	Instruction	Example	Meaning	Comments
Conditional branch	Branch if equal	`beq x5, x6, 100`	`if (x5 == x6) go to PC+100`	PC-relative branch if registers equal
	Branch if not equal	`bne x5, x6, 100`	`if (x5 != x6) go to PC+100`	PC-relative branch if registers not equal
	Branch if less than	`blt x5, x6, 100`	`if (x5 < x6) go to PC+100`	PC-relative branch if registers less
	Branch if greater or equal	`bge x5, x6, 100`	`if (x5 >= x6) go to PC+100`	PC-relative branch if registers greater or equal
	Branch if less, unsigned	`bltu x5, x6, 100`	`if (x5 < x6) go to PC+100`	PC-relative branch if registers less, unsigned
	Branch if greater or equal, unsigned	`bgeu x5, x6, 100`	`if (x5 >= x6) go to PC+100`	PC-relative branch if registers greater or equal, unsigned
Unconditional branch	Jump and link	`jal x1, 100`	`x1 = PC+4; go to PC+100`	PC-relative procedure call
	Jump and link register	`jalr x1, 100(x5)`	`x1 = PC+4; go to x5+100`	Procedure return; indirect call

Formato das instruções

- Organizadas em **seis classes**
  1. **Tipo-R** para **operações** de registradores
  2. **Tipo-I** para **valores imediatos curtos** 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**

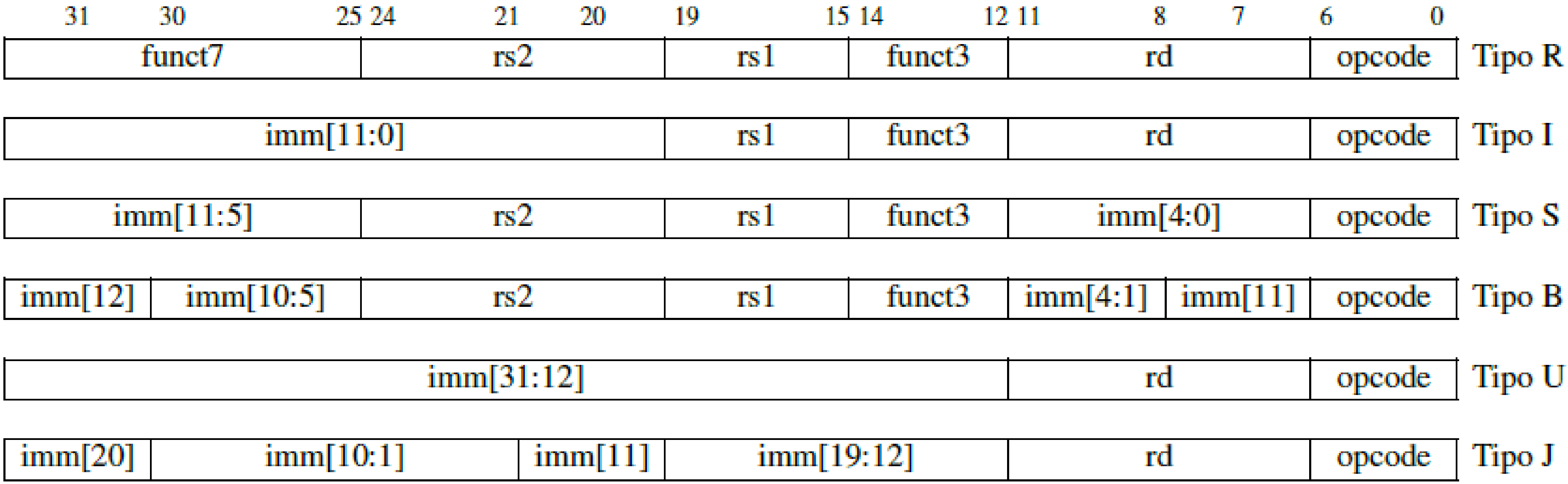


RISC-V Instructions	Name	Format
Add	add	R
Subtract	sub	R
Add immediate	addi	I
Load doubleword	ld	I
Store doubleword	sd	S
Load word	lw	I
Load word, unsigned	lwu	I
Store word	sw	S
Load halfword	lh	I
Load halfword, unsigned	lhu	I
Store halfword	sh	S
Load byte	lb	I
Load byte, unsigned	lbu	I
Store byte	sb	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

Formato das instruções

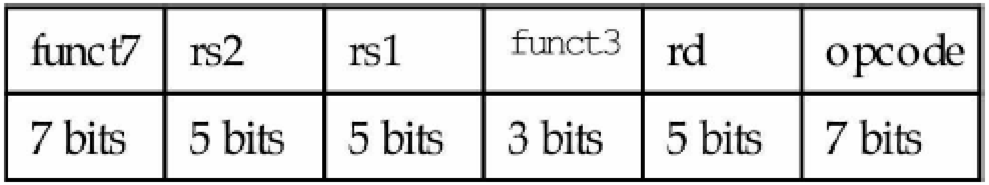


31	30 25	24 21	20	19 15	14 12	11 8	7	6 0	
funct7		rs2		rs1	funct3	rd		opcode	Tipo R
imm[11:0]				rs1	funct3	rd		opcode	Tipo I
imm[11:5]		rs2		rs1	funct3	imm[4:0]		opcode	Tipo S
imm[12]	imm[10:5]	rs2		rs1	funct3	imm[4:1]	imm[11]	opcode	Tipo B
imm[31:12]						rd		opcode	Tipo U
imm[20]	imm[10:1]		imm[11]	imm[19:12]		rd		opcode	Tipo J

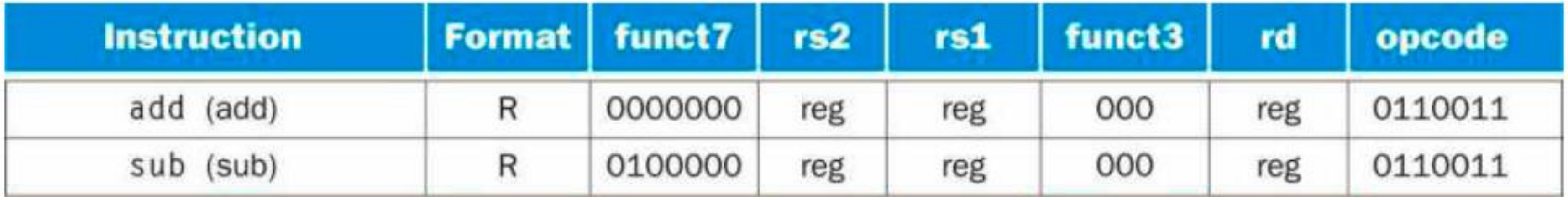
Formato R

funct7							rs2					rs1					funct3			rd					opcode						
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

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



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



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

– **Tipo-R** para **operações** de registradores

funct7							rs2					rs1					funct3			rd					opcode						
31	30	29	28	27	26	25	24	2 3	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0

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

imm[11:0]												rs1					funct3			rd					opcode						
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

– **Tipo-S** para **stores**

imm[11:5]							rs2					rs1					funct3			imm[4:0]					opcode						
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

– **Tipo-B** para **desvios condicionais**

<table>
<tr><th colspan="7">imm[12]|imm[10:5]</th><th colspan="5">rs2</th><th colspan="5">rs1</th><th colspan="3">funct3</th><th colspan="5">imm[4:1]|imm[11]</th><th colspan="7">opcode</th></tr>
<tr><td>31</td><td>30</td><td>29</td><td>28</td><td>27</td><td>26</td><td>25</td><td>24</td><td>23</td><td>22</td><td>21</td><td>20</td><td>19</td><td>18</td><td>17</td><td>16</td><td>15</td><td>14</td><td>13</td><td>12</td><td>11</td><td>10</td><td>9</td><td>8</td><td>7</td><td>6</td><td>5</td><td>4</td><td>3</td><td>2</td><td>1</td><td>0</td></tr>
</table>

– **Tipo-U** para **valores imediatos longos**

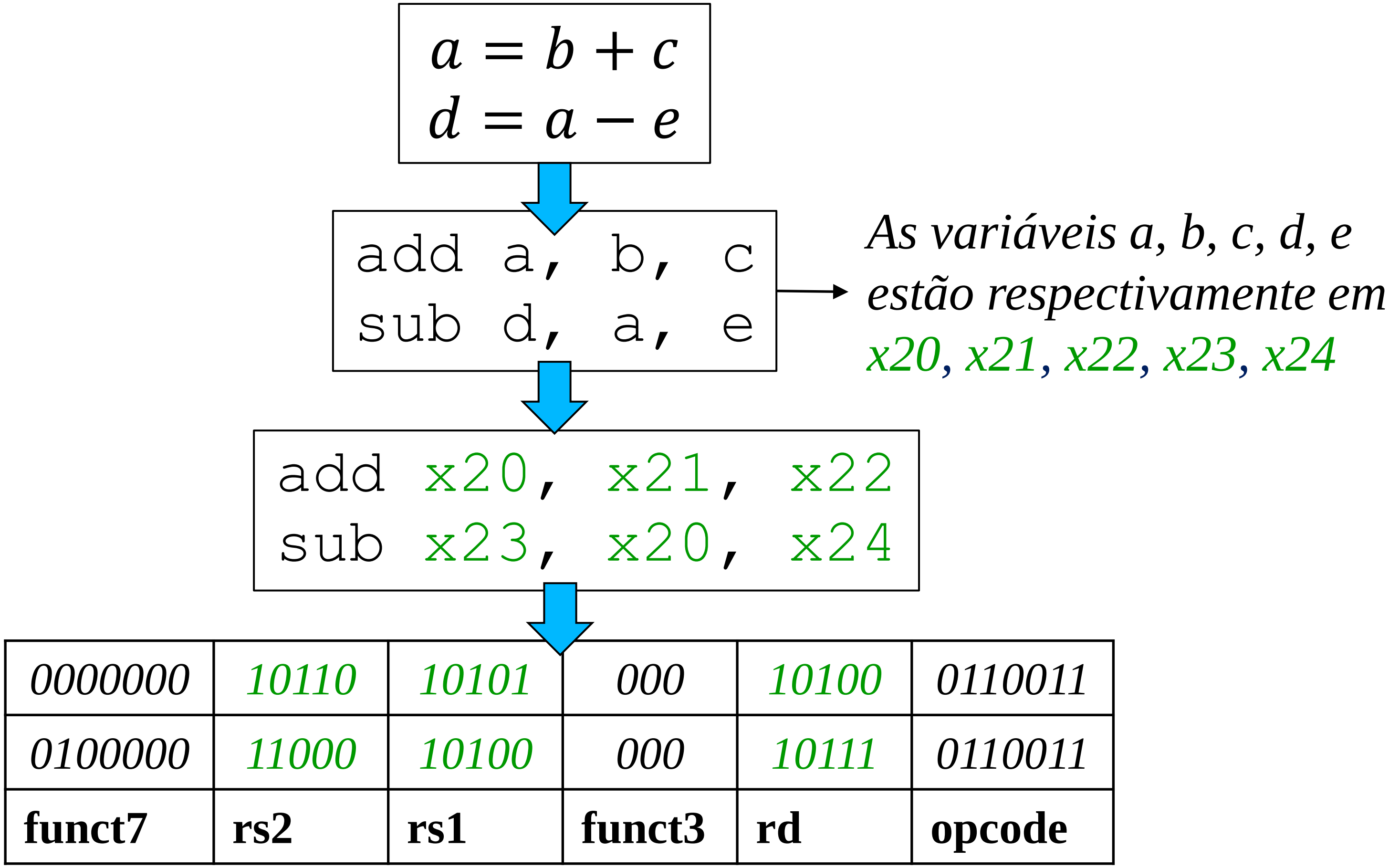
imm[31:12]																				rd					opcode						
31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0

– **Tipo-J** para **saltos incondicionais**

<table>
<tr><th colspan="20">imm[20] | imm[10:1] | imm[11] | imm[19:12]</th><th colspan="5">rd</th><th colspan="7">opcode</th></tr>
<tr><td>31</td><td>30</td><td>29</td><td>28</td><td>27</td><td>26</td><td>25</td><td>24</td><td>23</td><td>22</td><td>21</td><td>20</td><td>19</td><td>18</td><td>17</td><td>16</td><td>15</td><td>14</td><td>13</td><td>12</td><td>11</td><td>10</td><td>9</td><td>8</td><td>7</td><td>6</td><td>5</td><td>4</td><td>3</td><td>2</td><td>1</td><td>0</td></tr>
</table>

Exemplos

Exemplos



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

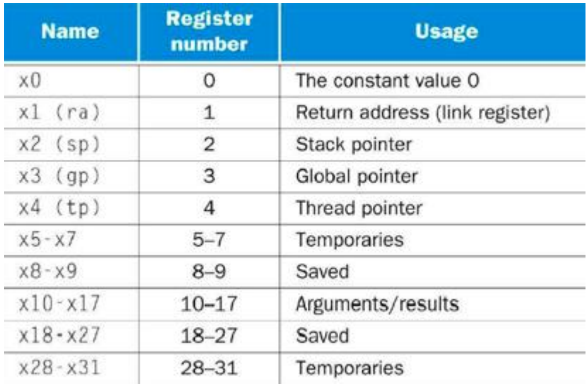
```
add a, b, c
sub d, a, e
```

As variáveis a, b, c, d, e estão respectivamente em x20, x21, x22, x23, x24

```
add x20, x21, x22
sub x23, x20, x24
```

0000000	*10110*	*10101*	*000*	*10100*	*0110011*
0100000	*11000*	*10100*	*000*	*10111*	*0110011*
funct7	**rs2**	**rs1**	**funct3**	**rd**	**opcode**

Exemplo 1



Name	Register number	Usage
x0	0	The constant value 0
x1 (ra)	1	Return address (link register)
x2 (sp)	2	Stack pointer
x3 (gp)	3	Global pointer
x4 (tp)	4	Thread pointer
x5-x7	5–7	Temporaries
x8-x9	8–9	Saved
x10-x17	10–17	Arguments/results
x18-x27	18–27	Saved
x28-x31	28–31	Temporaries

A[2] = A[2] + f;

- considere que a variável **f** está armazenada no registrador **x21** e que o segmento (base) do vetor **A** está em **x10**
- assuma que **A** é um vetor de dados de 32 bits; logo, **A[2]** indica *offset* de 2 palavras

- Código Assembly:

```
lw x9, 8(x10)       //x9 = A[2]
add x9, x9, x21     //x9 = A[2] + f
sw x9, 8(x10)       //A[2] = A[2] + f
```

```
lw rd, imm(rs1)
add rd, rs1, rs2
sw rs2, imm(rs1)
```

Exemplo 1

	opcode	funct3	funct7
lw	$(0000011)_2$ $(3)_{10}$	010	-
add	$(0110011)_2$ $(51)_{10}$	000	0000000
sw	$(0100011)_2$ $(35)_{10}$	010	-

A[2] = A[2] + f;

```
(I)lw x9, 8(x10)      //x9 = A[2]
(R)add x9, x21, x9    //x9 = A[2] + f
(S)sw x9, 8(x10)      //A[2] = A[2] + f
```

imm[11:0]		*rs1*	*funct3*	*rd*	*opcode*	(I)
funct7	*rs2*	*rs1*	*funct3*	*rd*	*opcode*	(R)
imm[11:5]	*rs2*	*rs1*	*funct3*	*imm[4:0]*	*opcode*	(S)

			2		3	(I)lw x9, 8(x10)
0			0		51	(R)add x9, x21, x9
			2		35	(S)sw x9, 8(x10)

Exemplo 1

	opcode	funct3	funct7
lw	$(0000011)_2$ $(3)_{10}$	010	-
add	$(0110011)_2$ $(51)_{10}$	000	0000000
sw	$(0100011)_2$ $(35)_{10}$	010	-

A[2] = A[2] + f;

```
(I)lw x9, 8(x10)    //x9 = A[2]
(R)add x9, x21, x9  //x9 = A[2] + f
(S)sw x9, 8(x10)    //A[2] = A[2] + f
```

imm[11:0]		*rs1*	*funct3*	*rd*	*opcode*	**(I)**
funct7	*rs2*	*rs1*	*funct3*	*rd*	*opcode*	**(R)**
imm[11:5]	*rs2*	*rs1*	*funct3*	*imm[4:0]*	*opcode*	**(S)**

8		*10*	*2*	*9*	*3*	`(I)lw x9, 8(x10)`
0	*9*	*21*	*0*	*9*	*51*	`(R)add x9, x21, x9`
0	*9*	*10*	*2*	*8*	*35*	`(S)sw x9, 8(x10)`

Exemplo 1

	opcode	funct3	funct7
lw	*$(0000011)_2$ $(3)_{10}$*	*010*	-
add	*$(0110011)_2$ $(51)_{10}$*	*000*	*0000000*
sw	*$(0100011)_2$ $(35)_{10}$*	*010*	-

A[2] = A[2] + f;

```
(I)lw x9, 8(x10)     //x9 = A[2]
(R)add x9, x21, x9   //x9 = A[2] + f
(S)sw x9, 8(x10)     //A[2] = A[2] + f
```

imm[11:0]		*rs1*	*funct3*	*rd*	*opcode*	(I)
funct7	*rs2*	*rs1*	*funct3*	*rd*	*opcode*	(R)
imm[11:5]	*rs2*	*rs1*	*funct3*	*imm[4:0]*	*opcode*	(S)

000000001000		*01010*	*010*	*01001*	*0000011*	`(I)lw x9, 8(x10)`
0000000	*01001*	*10101*	*000*	*01001*	*0110011*	`(R)add x9, x21, x9`
0000000	*01001*	*01010*	*010*	*01000*	*0100011*	`(S)sw x9, 8(x10)`

Exemplo 2

A[30] = h - A[30] + 1;

- considere que a variável **h** está armazenada no registrador **x21** e que o segmento (base) do vetor **A** está em **x10**
- assuma que A é um vetor de dados 32 bits; logo, **A[30]** indica *offset* de 30 palavras

- Código Assembly: ?

```
lw rd, imm(rs1)
sub rd, rs1, rs2
addi rd, rs1, imm
sw rs2, imm(rs1)
```

Exemplo 2

A[30] = h - A[30] + 1;

- considere que a variável **h** está armazenada no registrador **x21** e que o segmento (base) do vetor **A** está em **x10**
- assuma que A é um vetor de dados 32 bits; logo, **A[30]** indica *offset* de 30 palavras

- Código Assembly:

```
lw x9, 120(x10)    //x9=A[30]
sub x9, x21, x9    //x9=h-A[30]
addi x9, x9, 1     //x9=h-A[30]+1
sw x9, 120(x10)    //A[30]=h-A[30]+1
```

```
lw rd, imm(rs1)
sub rd, rs1, rs2
addi rd, rs1, imm
sw rs2, imm(rs1)
```

Exemplo 2

	opcode	funct3	funct7
lw	$(0000011)_2$ ou $(3)_{10}$	010	-
sub	$(0110011)_2$ ou $(51)_{10}$	000	0100000
addi	$(0010011)_2$ ou $(19)_{10}$	000	-
sw	$(0100011)_2$ ou $(35)_{10}$	010	-

A[30] = h - A[30] + 1;

```
(I)lw x9, 120(x10)    //x9=A[30]
(R)sub x9, x21, x9    //x9=h-A[30]
(I)addi x9, x9, 1     //x9=h-A[30]+1
(S)sw x9, 120(x10)    //A[30]=h-A[30]+1
```

imm[11:0]		rs1	funct3	rd	opcode	(I)
funct7	rs2	rs1	funct3	rd	opcode	(R)
imm[11:5]	rs2	rs1	funct3	imm[4:0]	opcode	(S)

Exemplo 2

	opcode	funct3	funct7
lw	$(0000011)_2$ ou $(3)_{10}$	010	-
sub	$(0110011)_2$ ou $(51)_{10}$	000	0100000
addi	$(0010011)_2$ ou $(19)_{10}$	000	-
sw	$(0100011)_2$ ou $(35)_{10}$	010	-

A[30] = h - A[30] + 1;

```
(I)lw x9, 120(x10)      //x9=A[30]
(R)sub x9, x21, x9      //x9=h-A[30]
(I)addi x9, x9, 1       //x9=h-A[30]+1
(S)sw x9, 120(x10)      //A[30]=h-A[30]+1
```

						(I)lw x9, 120(x10)
						(R)sub x9, x21, x9
						(I)addi x9, x9, 1
						(S)sw x9, 120(x10)

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.