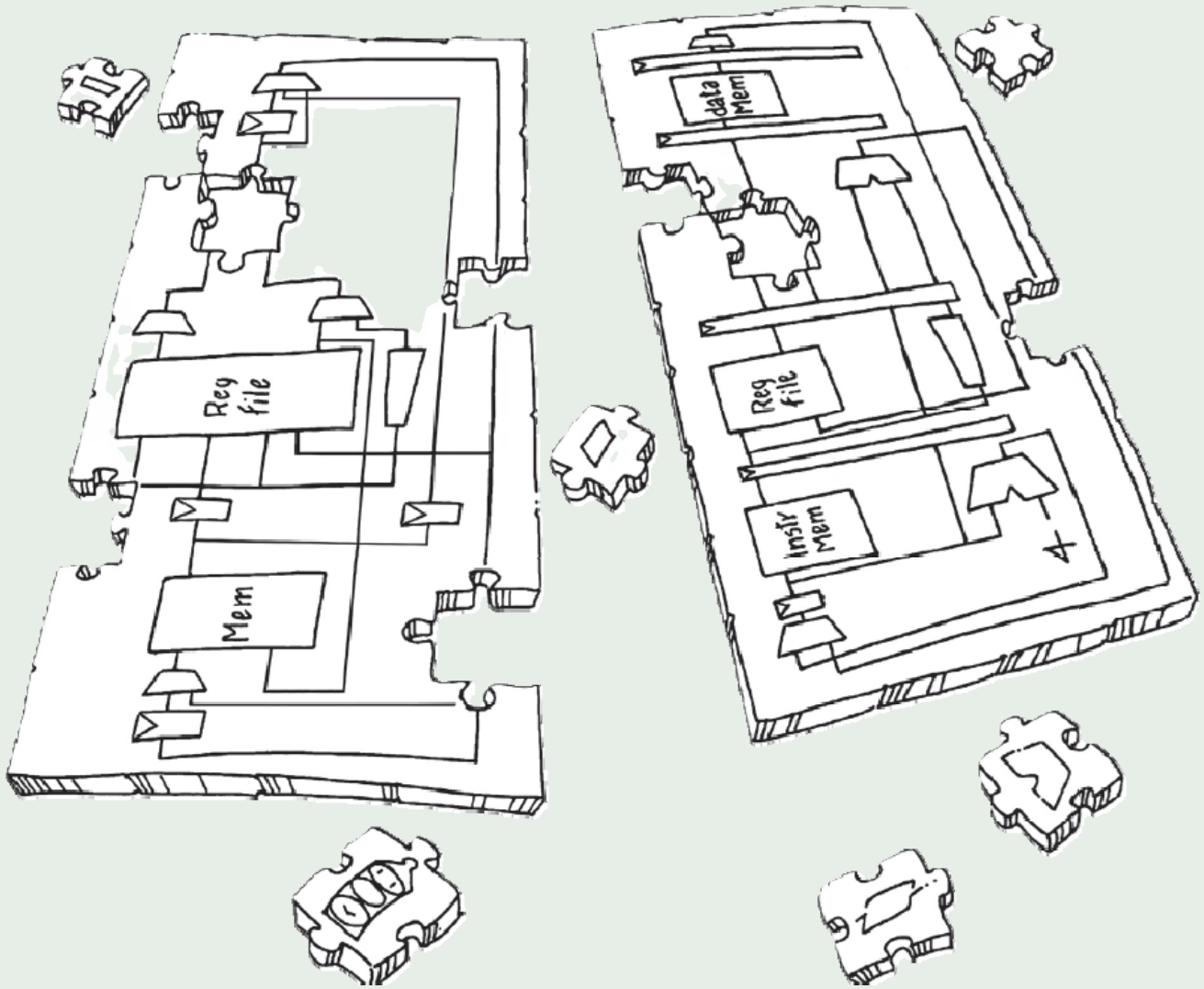


Fonte: Adaptado de Digital Design and Computer Architecture RISC-V Edition Sarah L. Harris David Harris

Arquitetura e Organização de Computadores 1

Prof. Luciano de Oliveira Neris
luciano@dc.ufscar.br

RISC - V

MIPS

- MIPS, acrônimo para Microprocessor without Interlocked Pipeline Stages (Microprocessador sem estágios interligados de pipeline)
- Surgiu nos anos 80 a partir de um trabalho realizado por John Hennessy, na Universidade de Stanford. O trabalho tinha como objetivo explorar o padrão RISC.
- O conceito introduzido por Hennessy foi tão bem sucedido que em 1984 foi formada a MIPS Technologies, Inc, a fim de comercializar os microprocessadores MIPS (R2000)

MIPS

- Quase 100 milhões de processadores MIPS fabricados em 2009. Esteve presente em diversos produtos: Utilizada pela NEC, Nintendo, Cisco, Silicon Graphics, Sony, impressoras HP e Fuji, etc.
- MIPS foi bastante utilizado para estudo da arquitetura de processadores por ser de notória simplicidade e clareza
- *Possui versões de 32 e 64 bits*
- *Continua sendo uma arquitetura importante, pois inspirou muitas das arquiteturas subsequentes como a RISC-V*

ARM

- Arquitetura RISC desenvolvida na mesma época que a arquitetura MIPS, na década de 1980. Originou-se na Acorn Computers Company que desenvolveu para BBC uma nova arquitetura para o projeto BBC Literacy.
- Uma nova companhia foi formada com a Acorn, VLSI e Apple Computers como sócios fundadores, conhecida como ARM Ltd (Advanced RISC Machine).
- Na última década ou mais, os processadores ARM dominaram o cenário dos dispositivos móveis devido à sua eficiência e um vasto ecossistema
- Adequado para uma variedade de sistemas de computação, desde microcontroladores até supercomputadores.

RISC-V

- "Reduced Instruction Set Computer V" (Pronuncia-se RISC-five)
- Conjunto de instruções aberto (open ISA Instruction Set Architecture).
- Pode ser utilizado para qualquer finalidade, por qualquer pessoa ou empresa, sem precisa pagar licenças ou direitos autorais (royalties)
- Projeto iniciado em 2010 na Universidade da California, em Berkeley, tornando-se depois em RISC-V Foundation em 2015 e finalmente RISC-V International em 2019, uma organização sem fins lucrativos na Suíca com mais de 3 mil membros participantes.
- Originalmente idealizada para ser utilizada na pesquisa e ensino da área de arquitetura de computadores, mas está se tornando um padrão de arquitetura aberta para a indústria.

RISC-V

- Projetado para processadores RISC custumizados para computações modernas, como aparelhos móveis, sistemas embarcados, internet das coisas, etc.
- Visa implementações de alto desempenho e baixo consumo de energia, com bases de 32, 64 e 128 bits.
- Adequado para uma variedade de sistemas de computação, desde microcontroladores até supercomputadores.
- Primeira ISA open source utilizada mundialmente.
- Mais de 10 bilhões de processadores RISC-V foram produzidos até 2023.

RISC-V

- A arquitetura RISC-V é baseada em registrador, ou seja, a CPU utiliza apenas registradores para realizar as suas operações aritméticas e lógicas. Não possui operações diretamente em memória.

- É uma arquitetura do tipo LOAD/STORE: Os valores tem que ser carregados nos registradores antes de qualquer operação

- As operações de acesso à memória só executam ou uma leitura da memória (load) ou uma escrita na memória (store)

- As instruções possuem poucos formatos e são do mesmo tamanho

RISC-V

- Utiliza nomes padronizados para descrever versões.

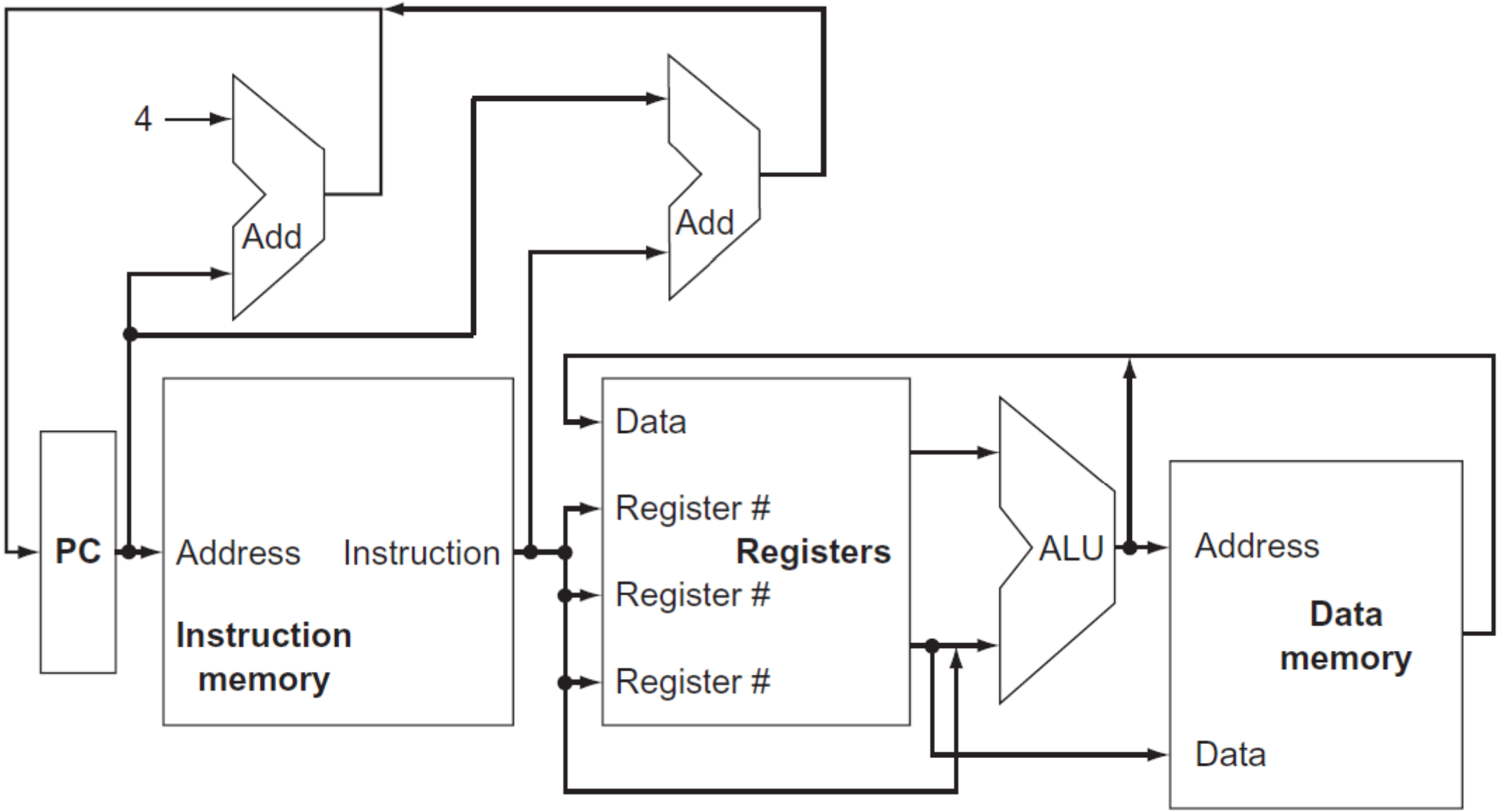
- RV[###][letras]
  - RV indica arquitetura RISC-V
  - ### indica o tamanho do registrador podendo ser:
    32, 64 ou 128
  - letras define uma extensão: I, M, A, F, D, G, C

  - Exemplos:
    - RV32I – The most basic RISC-V implementation
    - RV32IMAC – Integer + Multiply + Atomic + Compressed
    - RV64GC – 64bit IMAFDC
    - RV64GCXext – IMAFDC + a non-standard extension

Extension	Description
I	Integer
M	Integer Multiplication and Division
A	Atomics
F	Single-Precision Floating Point
D	Double-Precision Floating Point
G	General Purpose = IMAFD
C	16-bit Compressed Instructions
Non-Standard User-Level Extensions	
Xext	Non-standard extension “ext”

Common RISC-V Standard Extensions
*Not a complete list

RISC-V


4
Add
Add
PC
Address
Instruction
Instruction
memory
Data
Register #
Registers
Register #
Register #
ALU
Address
Data
memory
Data

RISC-V

- Características/restrições
  - Todas as instruções possuem 32 bits: nenhuma instrução utiliza apenas dois ou três bytes de memória e nenhuma instrução pode ser maior do que 4 bytes
  - Todas as instruções possuem opcode de 7 bits
  - Bi-endian

RISC-V: Registradores

- RV32I e RV64I tem 32 registradores de inteiros
- RV32E (E de Embarcado) tem 16 registradores de inteiros (para dispositivos embarcados com restrição de espaço)
- Extensões F e D tem 32 registradores de ponto flutuante
- ABI (Application Binary Interface) nomes padronizados para registradores.
- Adota-se uma convenção que especifica quais registradores devem ser utilizados em certas circunstâncias
- Os registradores PC (contador de programas) e IR (registrador de instrução) não fazem parte do banco de registradores

RISC-V: Registradores

Register	ABI Name	Description
x0	zero	Hard-wired zero
x1	ra	Return address
x2	sp	Stack pointer
x3	gp	Global pointer
x4	tp	Thread pointer
x5-7	t0-2	Temporaries
x8	s0/fp	Saved register/Frame pointer
x9	s1	Saved register
x10-11	a0-1	Function Arguments/return values
x12-17	a2-7	Function arguments
x18-27	s2-11	Saved registers
x28-31	t3-6	Temporaries

RISC-V: Modos

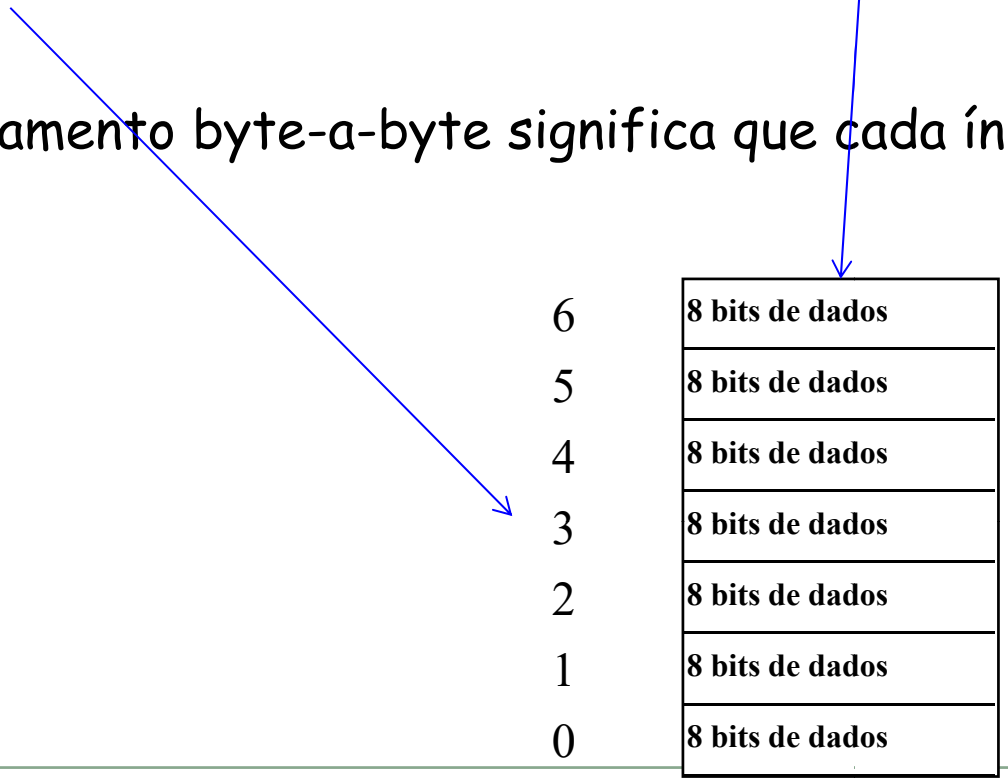
- Níveis de privilégio limitam o acesso à memória ou a instruções privilegiadas.
- Possui 4 níveis de privilégios chamados modos
- O modo Machine é o nível mais privilegiado

Nível	Nome do modo	Abreviatura
0	User/application	U
1	Supervisor	S
2	Hypervisor	H
3	Machine	M

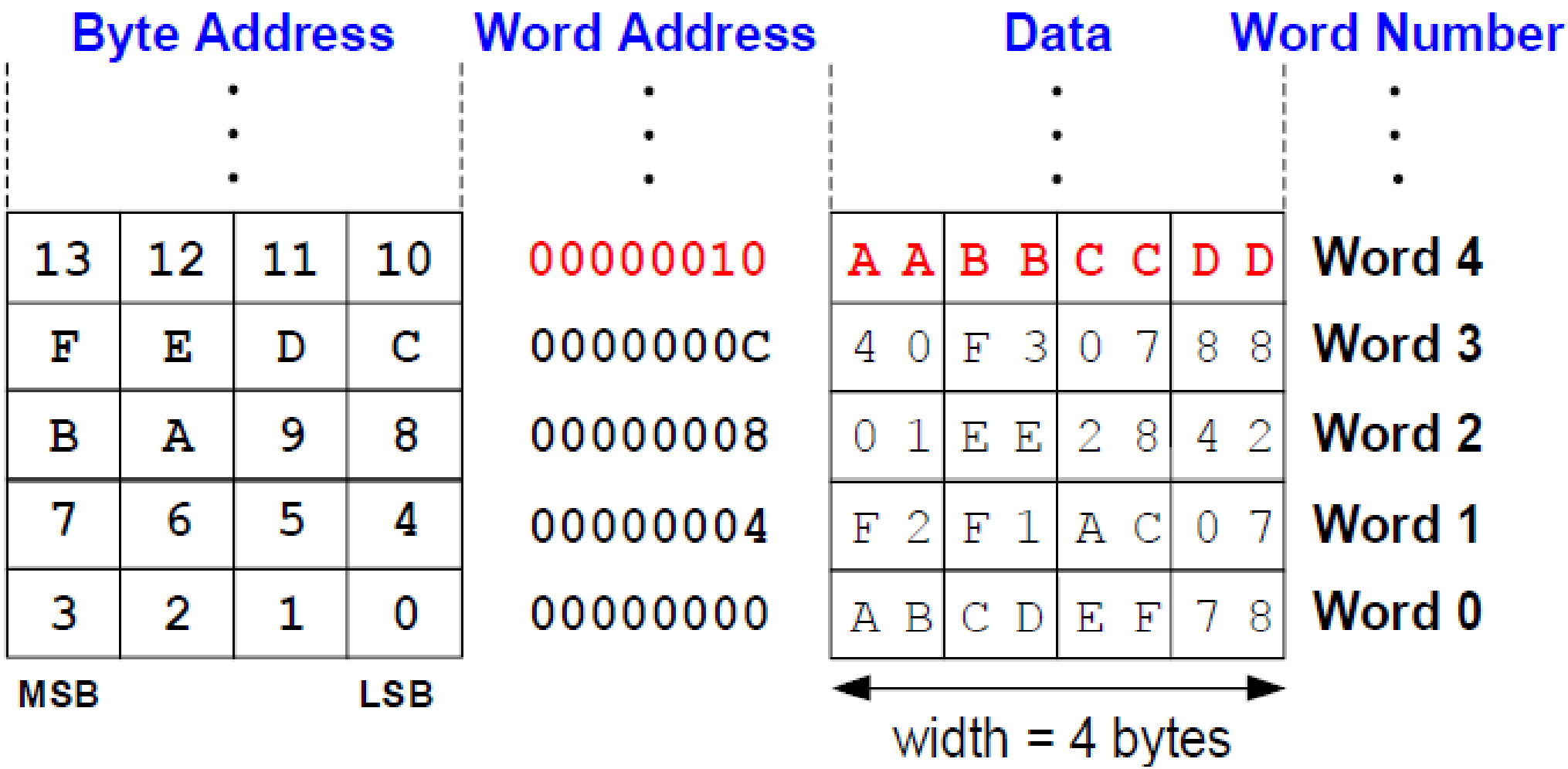
Organização da Memória

RISC-V: Memória

- A memória pode ser visa como um grande vetor de bytes, cada um com um endereço hexadecimal.
- Um endereço de memória é simplesmente um índice desse vetor.
- Endereçamento byte-a-byte significa que cada índice do vetor aponta para um byte único.



RISC-V: Memória



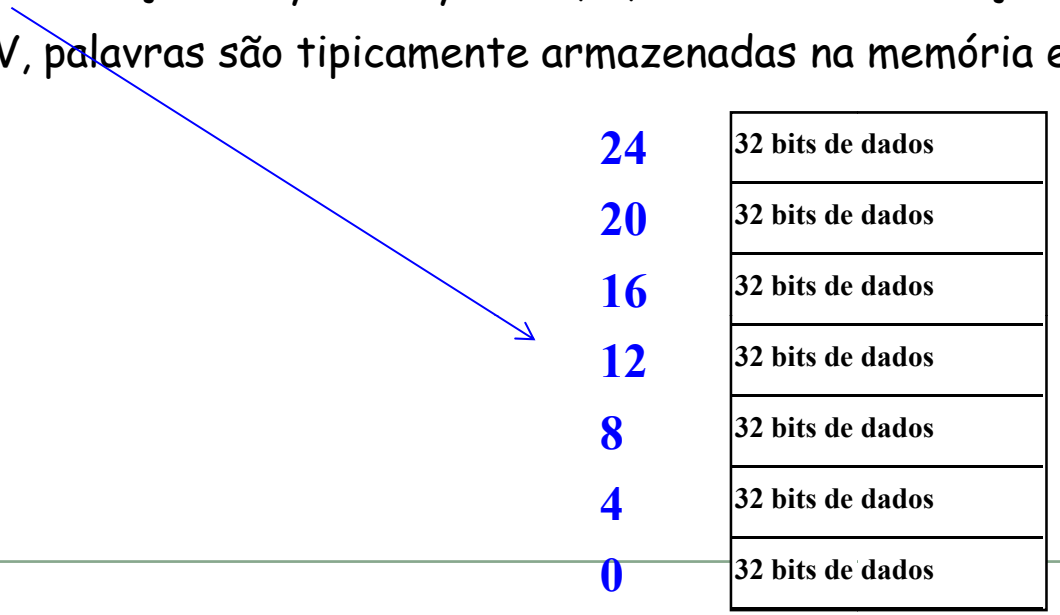
RISC-V: Memória

- A maior parte dos dados utilizados em programas são maiores do que bytes.
- RISC-V fornece as instruções para a leitura de dados da memória (load)
  - lw/lh/lb and sw/sh/sb
  - w= word (32 bits), h= half-word (16 bits), b= byte (8 bits)

2^{32} bytes endereçados byte-a-byte = 0, 1, 2….2^{32}-1 endereços = 4GByte

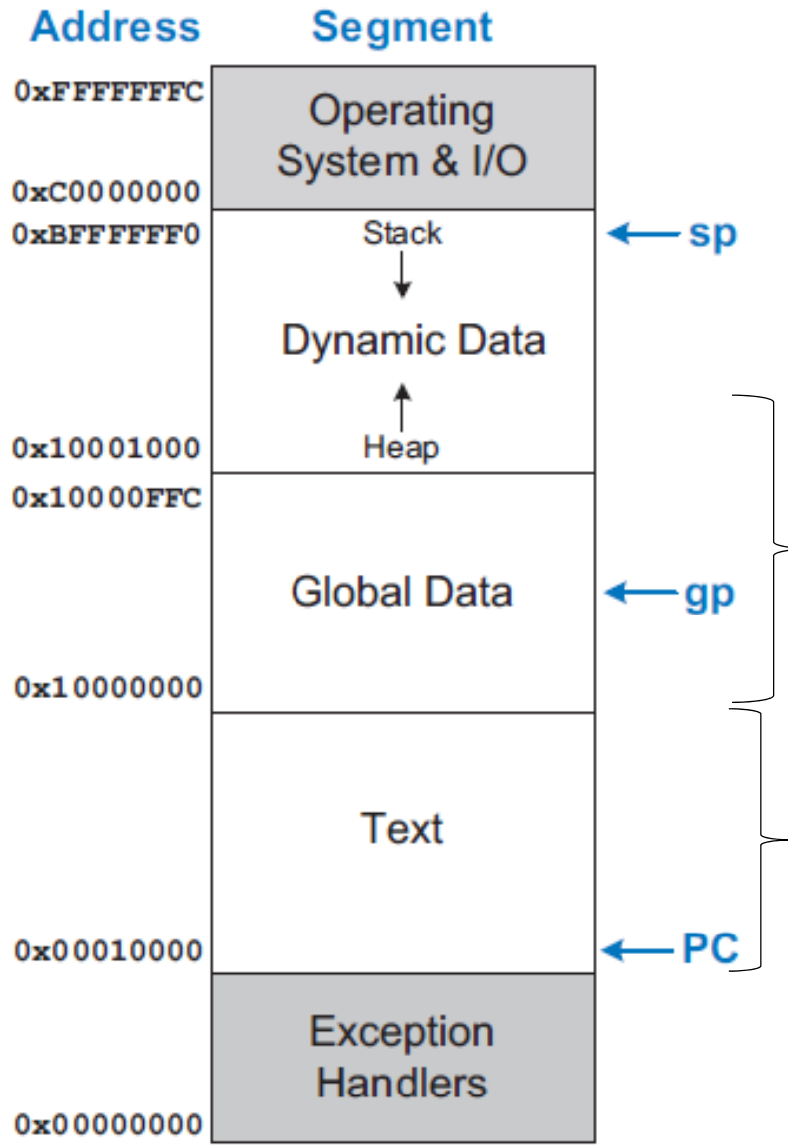
2^{30} words endereçados byte-a-byte = 0, 4, 8 … 2^{30}-4 endereços= 1GWord

Em RISC-V, palavras são tipicamente armazenadas na memória em LITTLE-ENDIAN. Palavras são alinhadas



RISC-V: Memória

Endereços de Memória (em Hex)



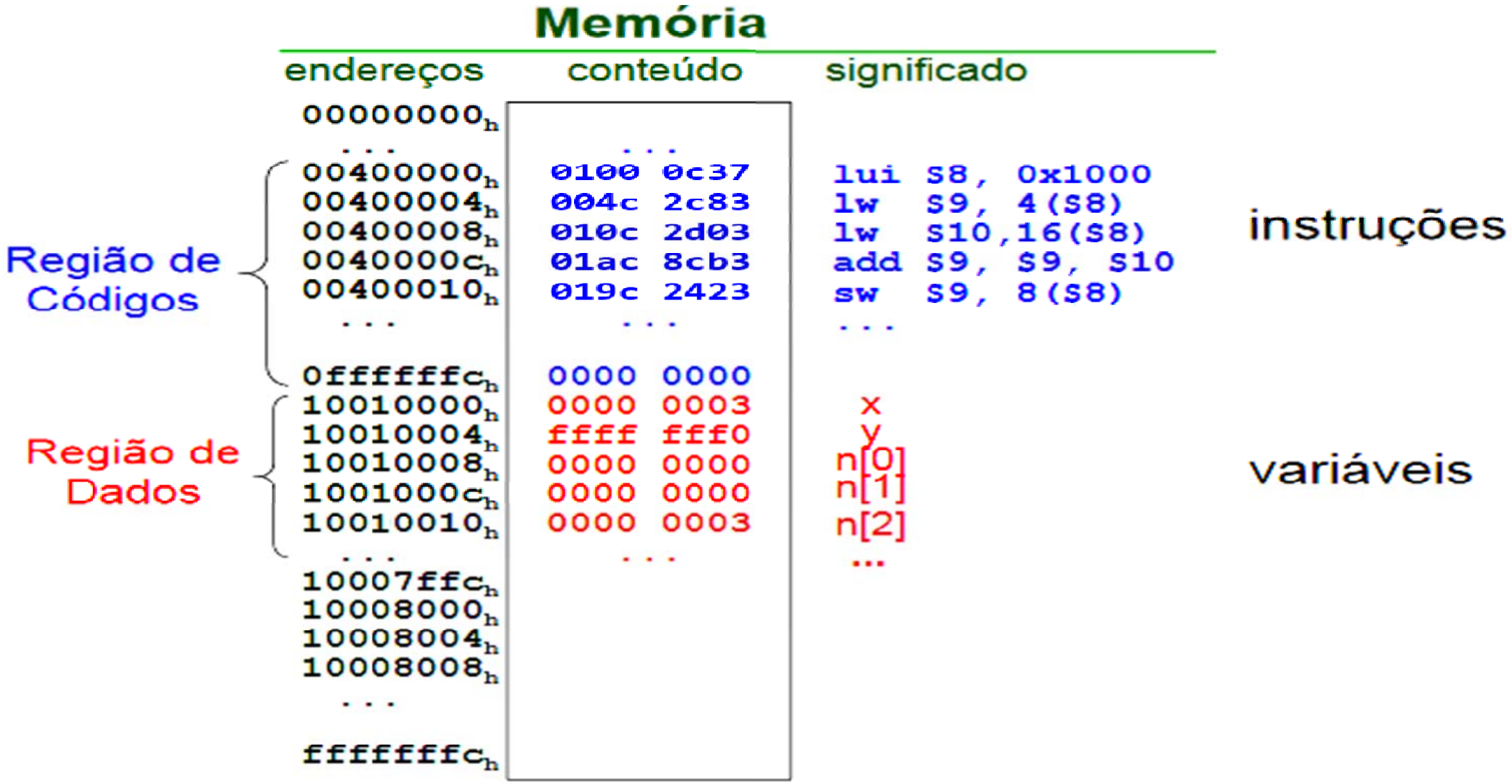
Pilha cresce "p/ baixo"

Segmento de Dados

Segmento de Código

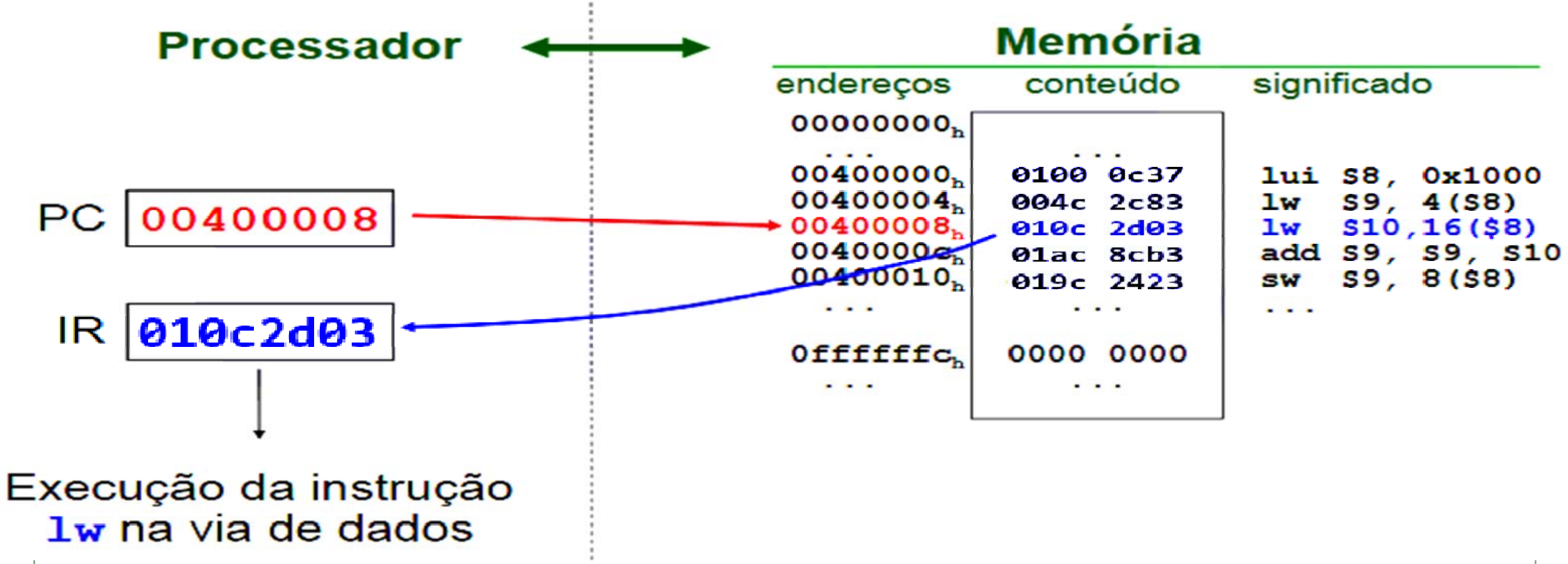
Figure 6.31 Example RISC-V memory map

RISC-V: Memória



Fonte: Adaptado de Slides do Prof. Frank Sill Torres DELT – Escola de Engenharia UFMG

RISC-V: Memória



Fonte: Adaptado de Slides do Prof. Frank Sill Torres DELT – Escola de Engenharia UFMG

Formato de Instruções

RISC-V: Formato de Instruções

- Instruções
  - Todas as instruções possuem 32 bits
  - Todas as instruções possuem opcode de 7 bits
  - O modo de endereçamento é codificado no opcode

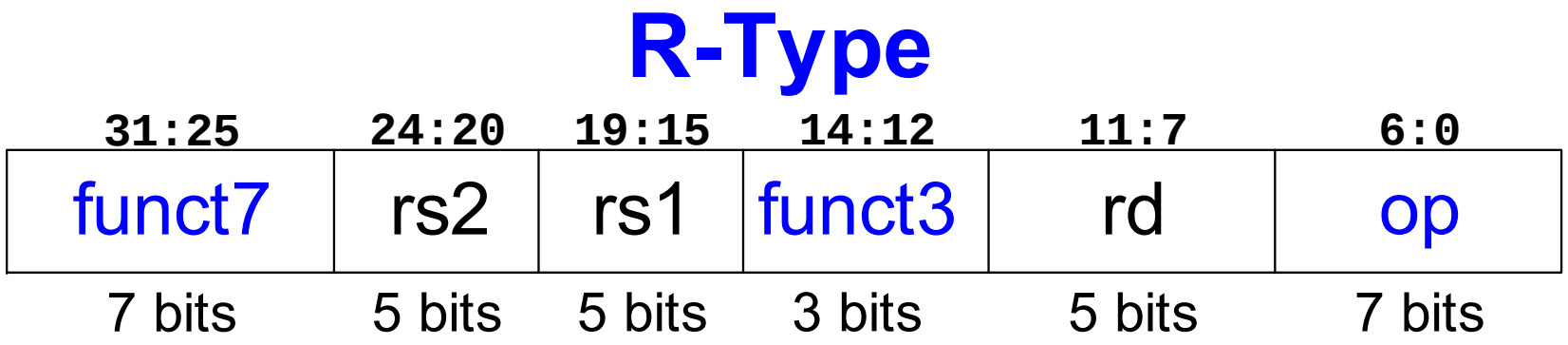
- As instruções podem ser classificadas em:
  - R-Type
  - I-Type
  - S/B Type
  - U/J Type

RISC-V: Formato de Instruções

7 bits	5 bits	5 bits	3 bits	5 bits	7 bits	
funct7	rs2	rs1	funct3	rd	op	**R-Type**
$imm_{11:0}$		rs1	funct3	rd	op	**I-Type**
$imm_{11:5}$	rs2	rs1	funct3	$imm_{4:0}$	op	**S-Type**
$imm_{12,10:5}$	rs2	rs1	funct3	$imm_{4:1,11}$	op	**B-Type**
$imm_{31:12}$				rd	op	**U-Type**
$imm_{20,10:1,11,19:12}$				rd	op	**J-Type**
20 bits				5 bits	7 bits	

RISC-V: Formato de Instruções

- R-Type (Tipo Registrador)
  - Três registradores operandos:
    - rs1, rs2: registradores de origem
    - rd: registrador de destino
  - Demais campos:
    - **op**: código da operação (opcode)
    - **funct7,funct3**: compoêm juntamente com o opcode a operação a ser realizada

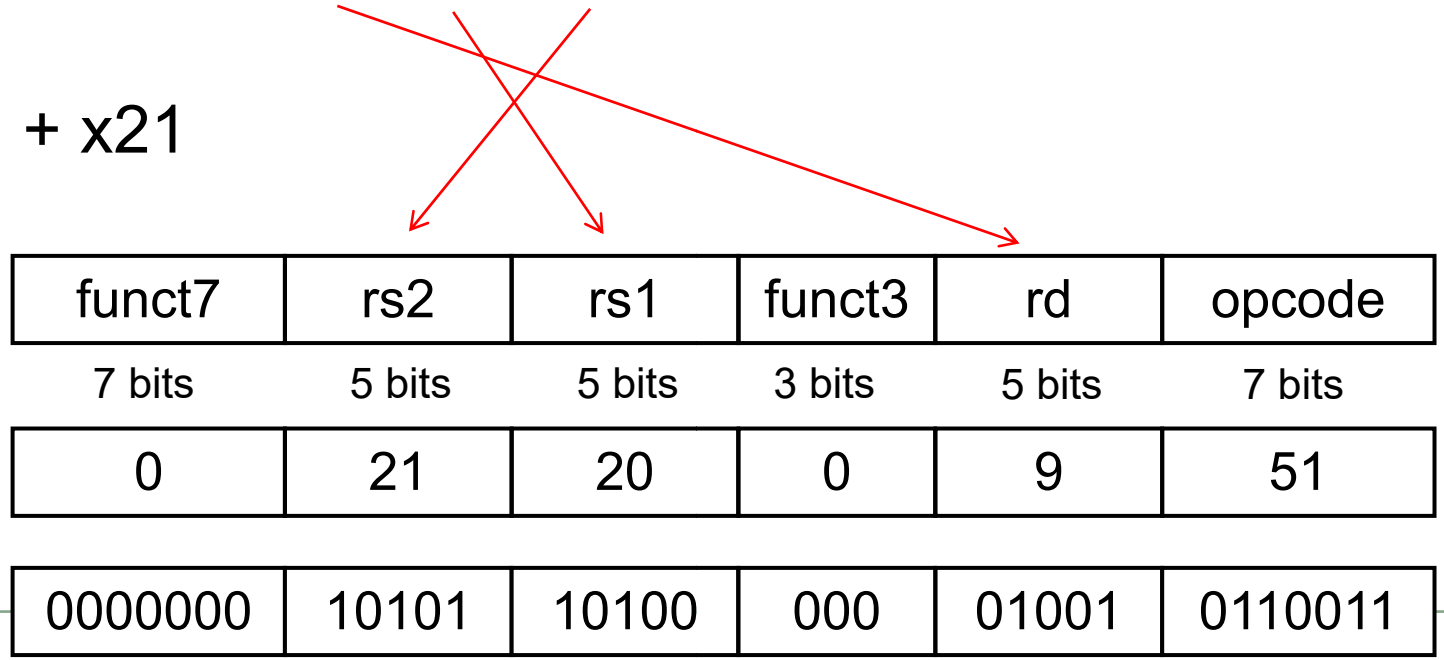


RISC-V: Formato de Instruções

- R-Type
  - Exemplo: add s1, s4, s5

    add x9, x20, x21

x9 = x20 + x21



$(015A04B3_{16})$

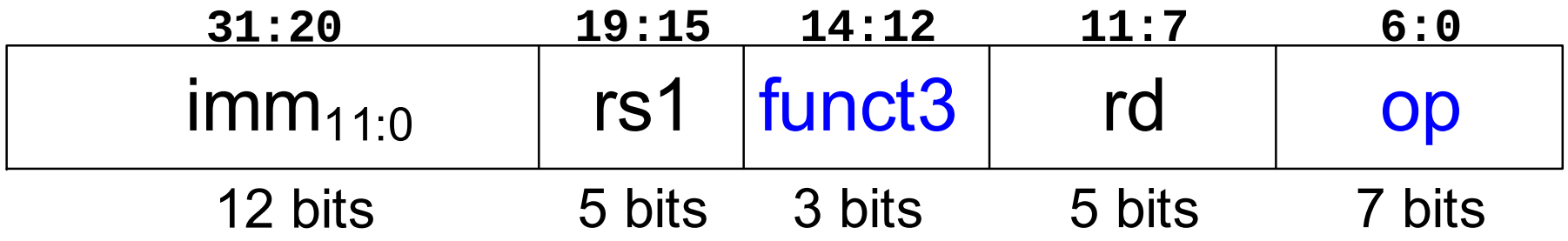
RISC-V: Formato de Instruções

- I-Type (Tipo Imediato)
  - Instruções envolvendo valor imediato
  - Um dos operandos pode ser representado na forma de um valor constante ou o endereço de palavra a ser acessada mantido dentro da própria instrução
  - Instruções típicas: aritméticas envolvendo constantes e de carga de dados

RISC-V: Formato de Instruções

- I-Type (Tipo Imediato)
  - Três operandos:
    - rs1: registrador de origem (endereço base ou valor)
    - rd: registrador de destino
    - imm: imediato em complemento de 2 (constante ou offset)
  - Demais campos:
    - **op**: código da operação (opcode)
    - **funct3**: compoêm juntamente com o opcode a operação a ser realizada

I-Type



RISC-V: Formato de Instruções

- I-Type:
  - Exemplos

rd = rs1 + imm

Assembly	$imm_{11:0}$	rs1	funct3	rd	op	Machine Code					
						$imm_{11:0}$	rs1	funct3	rd	op	
addi s0, s1, 12											
addi x8, x9, 12	12	9	0	8	19	0000 0000 1100	01001	000	01000	001 0011	(0x00C48413)
addi s2, t1, -14											
addi x18,x6, -14	-14	6	0	18	19	1111 1111 0010	00110	000	10010	001 0011	(0xFF230913)
lw t2, -6(s3)											
lw x7, -6(x19)	-6	19	2	7	3	1111 1111 1010	10011	010	00111	000 0011	(0xFFA9A383)
lh s1, 27(zero)											
lh x9, 27(x0)	27	0	1	9	3	0000 0001 1011	00000	001	01001	000 0011	(0x01B01483)
lb s4, 0x1F(s4)											
lb x20,0x1F(x20)	0x1F	20	0	20	3	0000 0001 1111	10100	000	10100	000 0011	(0x01FA0A03)
	12 bits	5 bits	3 bits	5 bits	7 bits	12 bits	5 bits	3 bits	5 bits	7 bits	

rd = rs1[imm]

RISC-V: Formato de Instruções

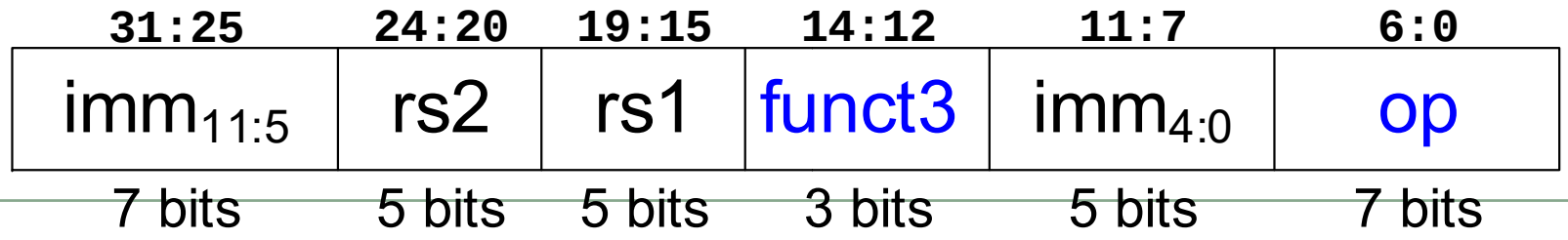
- S/B-Type: (Tipo Armazenamento / Tipo Salto Condcional)
  - Store-Type
  - Branch-Type
  - Difere somente na codificação do imediato

31:25	24:20	19:15	14:12	11:7	6:0	
$imm_{11:5}$	rs2	rs1	funct3	$imm_{4:0}$	op	**S-Type**
$imm_{12,10:5}$	rs2	rs1	funct3	$imm_{4:1,11}$	op	**B-Type**
7 bits	5 bits	5 bits	3 bits	5 bits	7 bits	

RISC-V: Formato de Instruções

- S-Type
  - Três operandos:
    - rs1: registrador base
    - rs2: registrador contendo o valor a ser armazenado na memória
    - imm: imediato em complemento de 2
  - Demais campos:
    - **op**: código da operação (opcode)
    - **funct3**: compoêm juntamente com o opcode a operação a ser realizada

S-Type



RISC-V: Formato de Instruções

- S-Type:
  - Exemplos

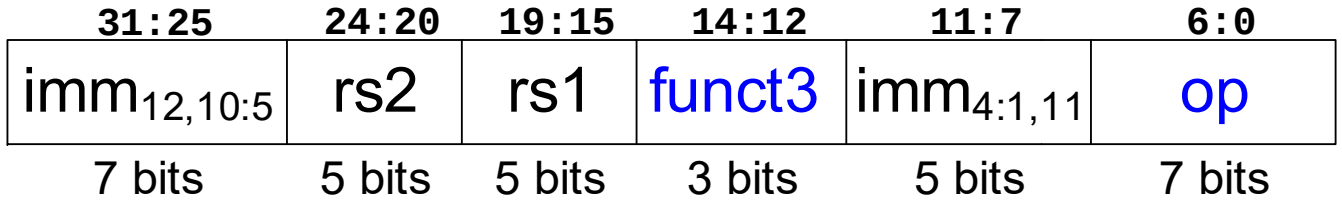
Mem[rs1 + imm] = rs2

Assembly	Field Values						Machine Code						
	$imm_{11:5}$	rs2	rs1	funct3	$imm_{4:0}$	op	$imm_{11:5}$	rs2	rs1	funct3	$imm_{4:0}$	op	
`sw t2, -6(s3)`	1111 111	7	19	2	11010	35	1111 111	00111	10011	010	11010	010 0011	**(0xFE79AD23)**
`sw x7, -6(x19)`													
`sh s4, 23(t0)`	0000 000	20	5	1	10111	35	0000 000	10100	00101	001	10111	010 0011	**(0x01429BA3)**
`sh x20,23(x5)`													
`sb t5, 0x2D(zero)`	0000 001	30	0	0	01101	35	0000 001	11110	00000	000	01101	010 0011	**(0x03E006A3)**
`sb x30,0x2D(x0)`	7 bits	5 bits	5 bits	3 bits	5 bits	7 bits	7 bits	5 bits	5 bits	3 bits	5 bits	7 bits	

RISC-V: Formato de Instruções

- B-Type
  - Três operandos:
    - rs1: registrador fonte 1
    - rs2: registrador fonte 2
    - $Imm_{12:11}$: imediato em complemento de 2 – endereço de offset
  - Demais campos:
    - **op**: código da operação (opcode)
    - **funct3**: compoêm juntamente com o opcode a operação a ser realizada

B-Type



RISC-V: Formato de Instruções

- B-Type:
  - Exemplos

```
# RISC-V Assembly
0x70       beq  s0, t5, L1
0x74       add  s1, s2, s3
0x78       sub  s5, s6, s7
0x7C       lw   t0, 0(s1)
0x80 L1:   addi s1, s1, -15
```

1 2 3 4

L1 is 4 instructions (i.e., **16 bytes**) past beq

$imm_{12:0}$ = 16 = 0 0 0 0 0 0 0 0 1 0 0 0 0

bit number = 12 11 10 9 8 7 6 5 4 3 2 1 0

Assembly **Field Values** **Machine Code**

```
beq s0, t5, L1
```

	$imm_{12,10:5}$	rs2	rs1	funct3	$imm_{4:1,11}$	op
Field Values	0000 000	30	8	0	1000 0	99
	7 bits	5 bits	5 bits	3 bits	5 bits	7 bits
Machine Code	0000 000	11110	01000	000	1000 0	110 0011

(0x01E40863)

RISC-V: Formato de Instruções

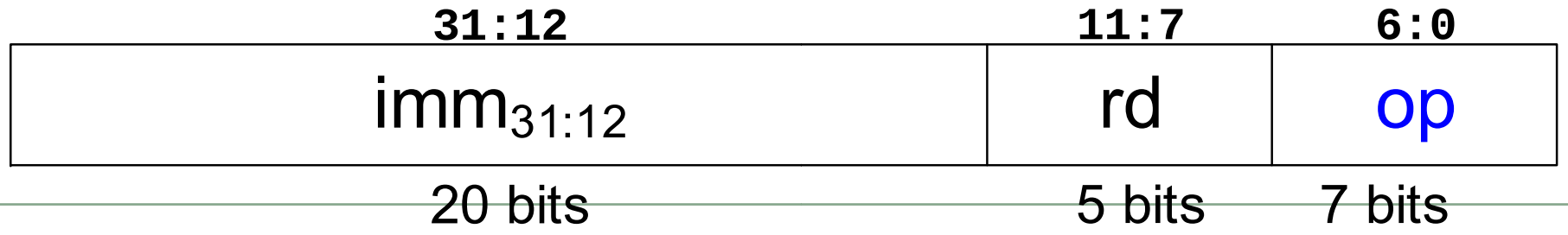
- U/J-Type: (Tipo Carga Alta / Tipo Salto Incondicional)
  - Upper-immediate-Type
  - Jump-Type
  - Difere somente na codificação do imediato

31:12	11:7	6:0	
$imm_{31:12}$	rd	op	**U-Type**
$imm_{20,10:1,11,19:12}$	rd	op	**J-Type**
20 bits	5 bits	7 bits	

RISC-V: Formato de Instruções

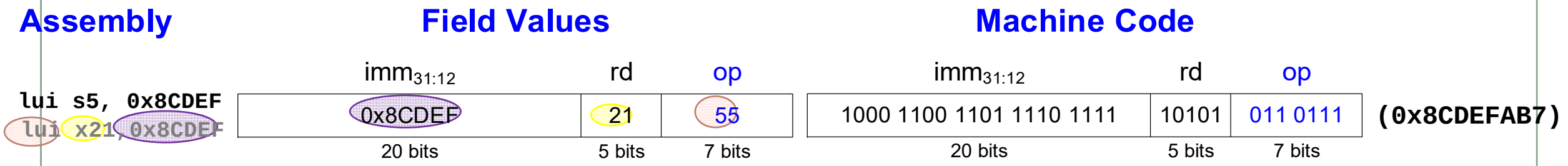
- U-Type
  - Usado na instrução load upper immediate (lui)
  - Dois operandos:
    - rd: registrador destino
    - $imm_{31:12}$: os 20 bits mais altos de um imediato de 32 bits
  - Demais campos:
    - **op**: código da operação (opcode)

U-Type



RISC-V: Formato de Instruções

- U-Type:
  - Exemplo



RISC-V: Formato de Instruções

- J-Type
  - Usado pela instrução jump-and-link (jal)
  - Dois operandos:
    - rd: registrador destino
    - $Imm_{20,10:1,11,19:12}$: 20 bits (20:1) de um imediato de 21 bits
  - Demais campos:
    - **op**: código da operação (opcode)

J-Type

RISC-V: Formato de Instruções

- J-Type:
  - Exemplos

```
# Address          RISC-V Assembly
0x0000540C         jal ra, func1
0x00005410         add s1, s2, s3
...                ...

0x000ABC04  func1: add s4, s5, s8
...                ...
```

func1 is 0xA67F8 bytes past **jal**

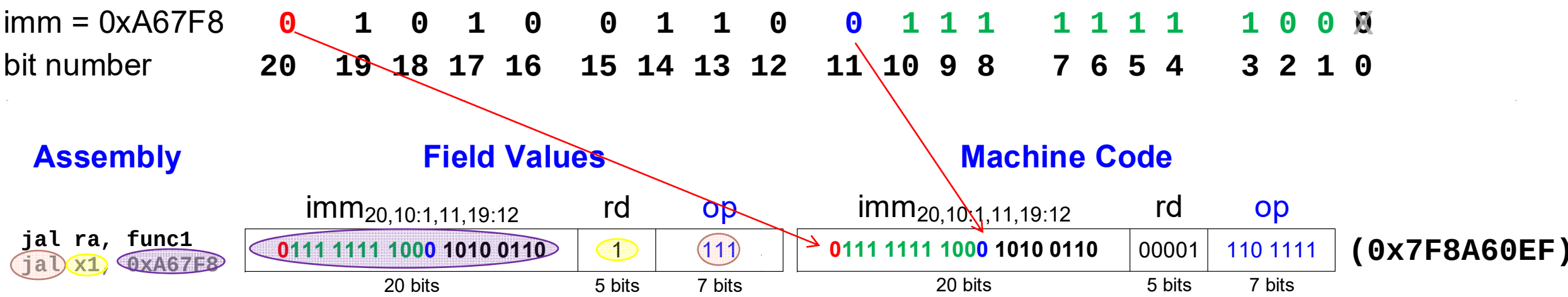


Table B.1 RV32I: RISC-V integer instructions

op	funct3	funct7	Type	Instruction	Description	Operation
0000011 (3)	000	–	I	`lb rd, imm(rs1)`	load byte	rd = SignExt([Address]$_{7:0}$)
0000011 (3)	001	–	I	`lh rd, imm(rs1)`	load half	rd = SignExt([Address]$_{15:0}$)
0000011 (3)	010	–	I	`lw rd, imm(rs1)`	load word	rd = [Address]$_{31:0}$
0000011 (3)	100	–	I	`lbu rd, imm(rs1)`	load byte unsigned	rd = ZeroExt([Address]$_{7:0}$)
0000011 (3)	101	–	I	`lhu rd, imm(rs1)`	load half unsigned	rd = ZeroExt([Address]$_{15:0}$)
0010011 (19)	000	–	I	`addi rd, rs1, imm`	add immediate	rd = rs1 + SignExt(imm)
0010011 (19)	001	0000000*	I	`slli rd, rs1, uimm`	shift left logical immediate	rd = rs1 << uimm
0010011 (19)	010	–	I	`slti rd, rs1, imm`	set less than immediate	rd = (rs1 < SignExt(imm))
0010011 (19)	011	–	I	`sltiu rd, rs1, imm`	set less than imm. unsigned	rd = (rs1 < SignExt(imm))
0010011 (19)	100	–	I	`xori rd, rs1, imm`	xor immediate	rd = rs1 ^ SignExt(imm)
0010011 (19)	101	0000000*	I	`srli rd, rs1, uimm`	shift right logical immediate	rd = rs1 >> uimm
0010011 (19)	101	0100000*	I	`srai rd, rs1, uimm`	shift right arithmetic imm.	rd = rs1 >>> uimm
0010011 (19)	110	–	I	`ori rd, rs1, imm`	or immediate	rd = rs1 | SignExt(imm)
0010011 (19)	111	–	I	`andi rd, rs1, imm`	and immediate	rd = rs1 & SignExt(imm)
0010111 (23)	–	–	U	`auipc rd, upimm`	add upper immediate to PC	rd = {upimm, 12'b0} + PC
0100011 (35)	000	–	S	`sb rs2, imm(rs1)`	store byte	[Address]$_{7:0}$ = rs2$_{7:0}$
0100011 (35)	001	–	S	`sh rs2, imm(rs1)`	store half	[Address]$_{15:0}$= rs2$_{15:0}$
0100011 (35)	010	–	S	`sw rs2, imm(rs1)`	store word	[Address]$_{31:0}$= rs2
0110011 (51)	000	0000000	R	`add rd, rs1, rs2`	add	rd = rs1 + rs2
0110011 (51)	000	0100000	R	`sub rd, rs1, rs2`	sub	rd = rs1 – rs2
0110011 (51)	001	0000000	R	`sll rd, rs1, rs2`	shift left logical	rd = rs1 << rs2$_{4:0}$
0110011 (51)	010	0000000	R	`slt rd, rs1, rs2`	set less than	rd = (rs1 < rs2)
0110011 (51)	011	0000000	R	`sltu rd, rs1, rs2`	set less than unsigned	rd = (rs1 < rs2)
0110011 (51)	100	0000000	R	`xor rd, rs1, rs2`	xor	rd = rs1 ^ rs2

RISC-V: Machine Codes

0110011 (51)	101	0000000	R	`srl rd, rs1, rs2`	shift right logical	`rd = rs1 >> rs2`$_{4:0}$
0110011 (51)	101	0100000	R	`sra rd, rs1, rs2`	shift right arithmetic	`rd = rs1 >>> rs2`$_{4:0}$
0110011 (51)	110	0000000	R	`or rd, rs1, rs2`	or	`rd = rs1 | rs2`
0110011 (51)	111	0000000	R	`and rd, rs1, rs2`	and	`rd = rs1 & rs2`
0110111 (55)	–	–	U	`lui rd, upimm`	load upper immediate	`rd = {upimm, 12'b0}`
1100011 (99)	000	–	B	`beq rs1, rs2, label`	branch if =	`if (rs1 == rs2) PC = BTA`
1100011 (99)	001	–	B	`bne rs1, rs2, label`	branch if ≠	`if (rs1 ≠ rs2) PC = BTA`
1100011 (99)	100	–	B	`blt rs1, rs2, label`	branch if <	`if (rs1 < rs2) PC = BTA`
1100011 (99)	101	–	B	`bge rs1, rs2, label`	branch if ≥	`if (rs1 ≥ rs2) PC = BTA`
1100011 (99)	110	–	B	`bltu rs1, rs2, label`	branch if < unsigned	`if (rs1 < rs2) PC = BTA`
1100011 (99)	111	–	B	`bgeu rs1, rs2, label`	branch if ≥ unsigned	`if (rs1 ≥ rs2) PC = BTA`
1100111 (103)	000	–	I	`jalr rd, rs1, imm`	jump and link register	`PC = rs1 + SignExt(imm), rd = PC + 4`
1101111 (111)	–	–	J	`jal rd, label`	jump and link	`PC = JTA, rd = PC + 4`

*Encoded in instr$_{31:25}$, the upper seven bits of the immediate field

RISC-V: Pseudo instruções

Pseudoinstruction	RISC-V Instructions	Description	Operation
`nop`	`addi x0, x0, 0`	no operation	
`li rd, imm11:0`	`addi rd, x0, imm11:0`	load 12-bit immediate	`rd = SignExtend(imm11:0)`
`li rd, imm31:0`	`lui rd, imm31:12*` `addi rd, rd, imm11:0`	load 32-bit immediate	`rd = imm31:0`
`mv rd, rs1`	`addi rd, rs1, 0`	move (also called "register copy")	`rd = rs1`
`not rd, rs1`	`xori rd, rs1, −1`	one's complement	`rd = ~rs1`
`neg rd, rs1`	`sub rd, x0, rs1`	two's complement	`rd = −rs1`
`seqz rd, rs1`	`sltiu rd, rs1, 1`	set if = 0	`rd = (rs1 == 0)`
`snez rd, rs1`	`sltu rd, x0, rs1`	set if ≠ 0	`rd = (rs1 ≠ 0)`
`sltz rd, rs1`	`slt rd, rs1, x0`	set if < 0	`rd = (rs1 < 0)`
`sgtz rd, rs1`	`slt rd, x0, rs1`	set if > 0	`rd = (rs1 > 0)`
`beqz rs1, label`	`beq rs1, x0, label`	branch if = 0	`if (rs1 == 0) PC = label`
`bnez rs1, label`	`bne rs1, x0, label`	branch if ≠ 0	`if (rs1 ≠ 0) PC = label`
`blez rs1, label`	`bge x0, rs1, label`	branch if ≤ 0	`if (rs1 ≤ 0) PC = label`
`bgez rs1, label`	`bge rs1, x0, label`	branch if ≥ 0	`if (rs1 ≥ 0) PC = label`
`bltz rs1, label`	`blt rs1, x0, label`	branch if < 0	`if (rs1 < 0) PC = label`
`bgtz rs1, label`	`blt x0, rs1, label`	branch if > 0	`if (rs1 > 0) PC = label`
`ble rs1, rs2, label`	`bge rs2, rs1, label`	branch if ≤	`if (rs1 ≤ rs2) PC = label`
`bgt rs1, rs2, label`	`blt rs2, rs1, label`	branch if >	`if (rs1 > rs2) PC = label`
`bleu rs1, rs2, label`	`bgeu rs2, rs1, label`	branch if ≤ (unsigned)	`if (rs1 ≤ rs2) PC = label`
`bgtu rs1, rs2, label`	`bltu rs2, rs1, offset`	branch if > (unsigned)	`if (rs1 > rs2) PC = label`
`j label`	`jal x0, label`	jump	`PC = label`
`jal label`	`jal ra, label`	jump and link	`PC = label, ra = PC + 4`
`jr rs1`	`jalr x0, rs1, 0`	jump register	`PC = rs1`
`jalr rs1`	`jalr ra, rs1, 0`	jump and link register	`PC = rs1, ra = PC + 4`
`ret`	`jalr x0, ra, 0`	return from function	`PC = ra`
`call label`	`jal ra, label`	call nearby function	`PC = label, ra = PC + 4`
`call label`	`auipc ra, offset31:12*` `jalr ra, ra, offset11:0`	call far away function	`PC = PC + offset, ra = PC + 4`
`la rd, symbol`	`auipc rd, symbol31:12*` `addi rd, rd, symbol11:0`	load address of global variable	`rd = PC + symbol`

RISC-V: Princípios de Projeto

- Simplicidade favorece a regularidade
  - Instruções fixas de 32 bits
  - Pequeno número de formato de instruções
  - O opcode está sempre nos 7 bits menos significativos
- Bons projetos requerem compromissos
  - 4 formatos de instruções
- Menor é melhor
  - Conjunto de instruções limitado
  - Número de registradores no banco de registradores limitado
  - Número de modos de endereçamento limitado
- Torne o caso comum rápido
  - Operandos aritméticos estão no banco de registradores (máquina load-store)
  - Permitir que as instruções tenham operandos imediatos

Instruções

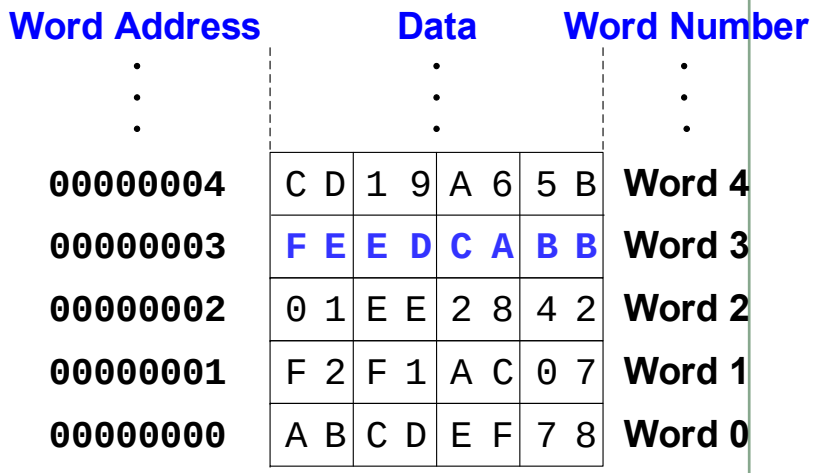
RISC-V: Instruções

- Transferência de Dados
  - Como carregar dados da memória (load)?
  - **Formato**:
    - **lw t1, 5(s0)**
    - **lw destination, offset(base)**
  - **Cálculo do endereço:**
    - Adicionar o endereço base (s0) ao *offset* (5)
    - Endereço = (s0 + 5)
  - **Resultado:**
    - t1 armazena o valor contido no endereço de memória (s0 + 5)

Qualquer registrador pode ser utilizado como endereço base

RISC-V: Instruções

- Transferência de Dados
  - Como armazenar dados na memória (store)?
  - **Formato**:
    - **sw t4, 3(zero)**
    - **sw origem, offset(base)**
  - **Cálculo do endereço:**
    - Adicionar o endereço base (zero) ao *offset* (3)
    - Endereço = (0 + 3)
  - **Resultado:**
    - O conteúdo de t4 é armazenado no endereço de memória (0 + 3)



Word Address	Data	Word Number
00000004	C D 1 9 A 6 5 B	Word 4
00000003	F E E D C A B B	Word 3
00000002	0 1 E E 2 8 4 2	Word 2
00000001	F 2 F 1 A C 0 7	Word 1
00000000	A B C D E F 7 8	Word 0

Qualquer registrador pode ser utilizado como endereço base

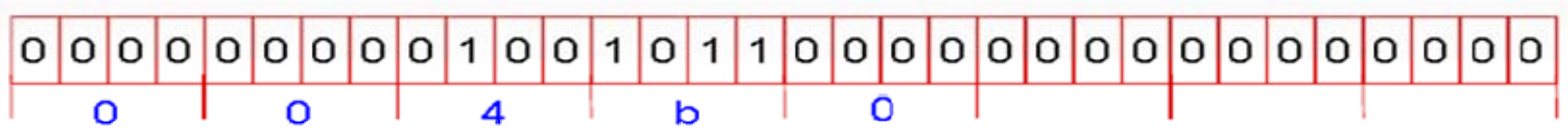
RISC-V: Instruções

- Transferência de Dados
  - Como carregar constantes de 32 bits?
    - Utilizar a instrução lui (*load upper immediate*) para os MSBs
    - Utilizar a instrução addi (*add immediate*) para os LSBs

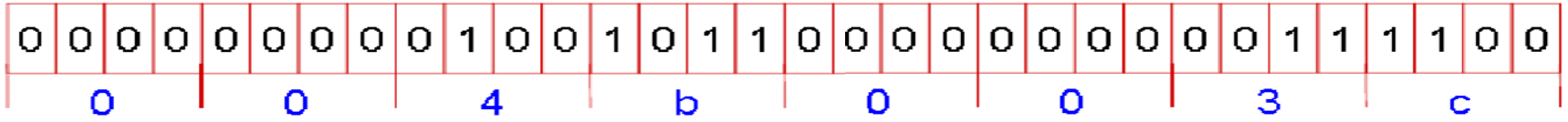
RISC-V: Instruções

- Transferência de Dados
  - Ex: t0 = 0x004b003c

lui t0, 0x004B0



addi t0,t0, 0x03C



RISC-V: Instruções

- Desvio condicional (branch)
  - Em linguagens de programação de alto nível o comando de tomada de decisão é geralmente implementado através do comando if
  - O MIPS possui duas principas instruções de desvio:
    - beq r1, r2, L1 (branch if equal)
      - se os valores de r1 e r2 são iguais, desvia para a linha identificada pelo label L1
    - bne r1, r2, L1 (branch if not equal)
      - se os valores de r1 e r2 são diferentes, desvia para a linha identificada pelo label L1

if cond
then goto label

RISC-V: Instruções

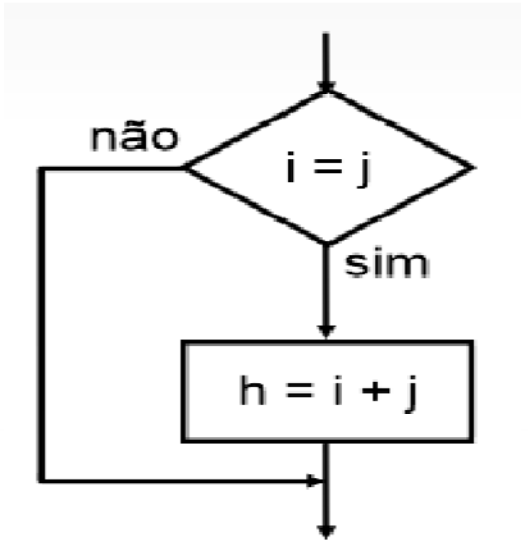
- Desvio condicional (branch)

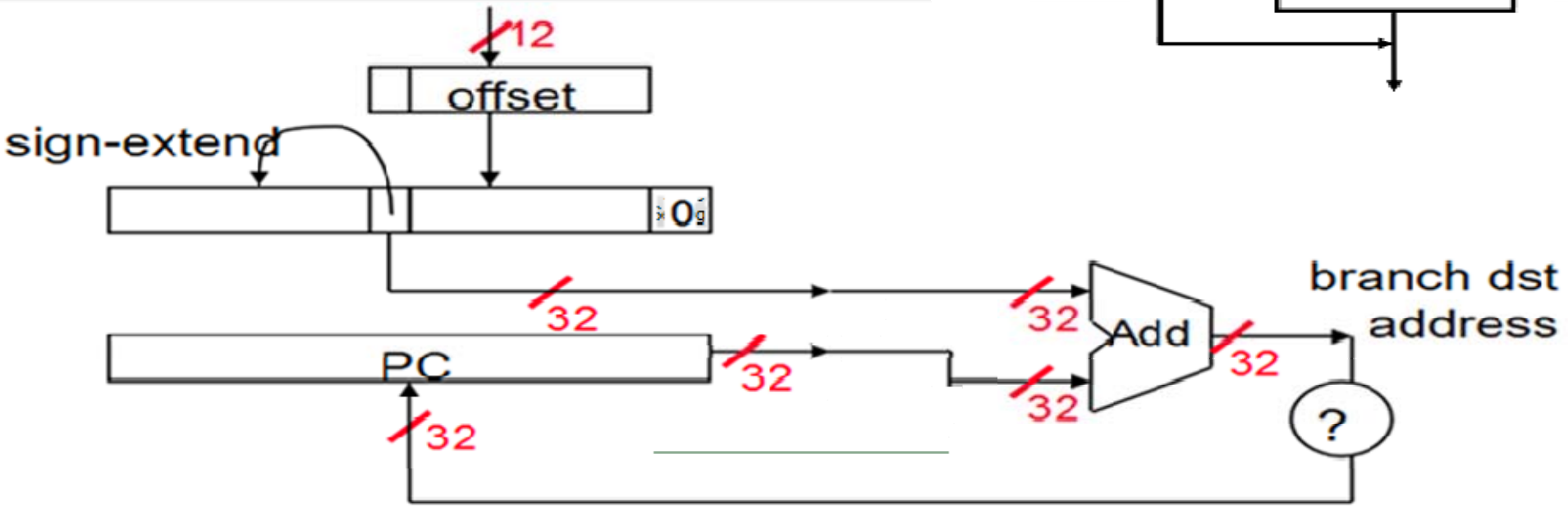
```
bge  s0, s1, label      #if s0>=s1 goto label

ble  s0, s1, label      #if s0<=s1 goto label

blt  s0, s1, label      #if s0<s1 goto label

beq  s0, s1, label      #if s0==s1 goto label

bne  s0, s1, label      #if s0!=s1 goto label
```

RISC-V: Instruções

- Desvio condicional (branch)

```
    bne s8, s9, exit
    add s10, s8, s9
exit: ...
```

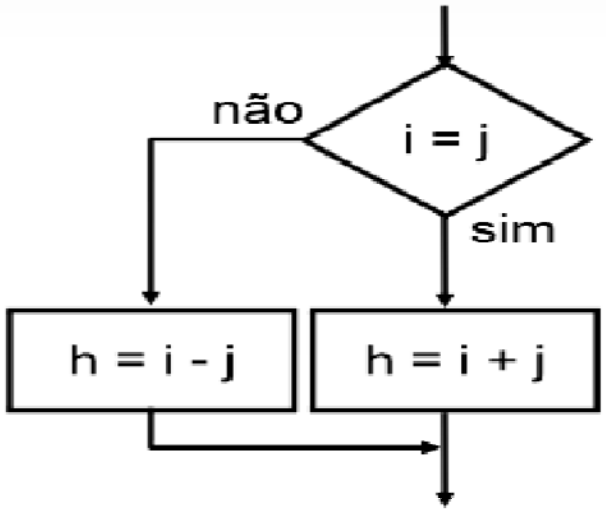




RISC-V: Instruções

- Desvio condicional (branch)

```
      bne  s8, s9, else
      add  s10, s8, s9
      j exit
else: sub  s10, s8, s9
exit : ...
```



RISC-V: Instruções

- Instrução de Comparação
  - Permite testar se uma variável é menor do que outra
  - Compara o conteúdo de dois registradores e atribui o valor 1 a um terceiro registrador se primeiro < segundo. Caso contrário, é atribuído o valor 0 ao terceiro registrador.
  - slt (set on less than)

```
slt t0, s3, s4          # t0 recebe 1 se s3 < s4
                        # t0 recebe 0 se s3 ≥ s4
```

RISC-V: Instruções

- Instruções de Controle
  - Com instruções simples como Branch e Jump é possível construir estruturas como:
    - If-Then-Else
    - Loops (For, While, Repeat Until)
    - Chamadas de funções

RISC-V: Instruções

- Instruções de Controle

```
if (t0==t1)
then
   t2= 25
else
   t2= 77
t3 = t3 + t2
```

```
            beq t0, t1, blockA
            j blocoB
blocoA:     li t2, 25
            j exit
blocoB:     li t2, 77
exit:       addu t3, t3, t2
```

RISC-V: Instruções

- Instruções de Controle

```
repeat … until t0 > t1
  t0 = t0 + 1
```

```
loop1:
        addi t0, t0, 1
        ble t0, t1, loop1    # if t0 <= t1 goto loop1
```

RISC-V: Instruções

- Instruções de Controle

```
switch (k)
{
   case 0:
            f = i + j;
            break;
   case 1:
            f = g + h;
            break;
   case 2:
            f = g - h;
            break;
   case 3: f = i - j;
}
```

```
            slt t3, s5, zero
            bne t3, zero, Exit
            slt t3, s5, t2
            beq t3, zero, Exit
            add t1, s5, s5
            add t1, t1, t1
            add t1, t1, t4
            lw t0, 0 (t1)
            jr t0
L0:         add s0, s3, s4
            j Exit
L1:         add s0, s1, s2
            j Exit
L2:         sub s0, s1, s2
            j Exit
L3:         sub s0, s3, s4
Exit:
```

RISC-V: Instruções

- Instruções de Controle

```
// add the numbers from 0 to 9
int sum = 0;
int i;

for (i=0; i!=10; i = i+1)
{
  sum = sum + i;
}
```

```
# s0 = i, s1 = sum
        addi s1, zero, 0
        add  s0, zero, zero
        addi t0, zero, 10
for:
        beq  s0, t0, done
        add  s1, s1, s0
        addi s0, s0, 1
        j    for
done:
```

Sub-rotina

RISC-V: Sub-rotina

- Passos pra a execução
  1. Passagem de parâmetros
  2. Transferência do controle de execução
  3. Aquisição dos recursos de armazenamento necessários para o procedimento
  4. Realização da tarefa desejada.
  5. Armazenamento do resultado em local acessível ao programa que chamou a sub-rotina.
  6. Retorno à execução do programa original.

RISC-V: Sub-rotina

- Convenção
  - a0 – a1: dois registradores para passagem de parâmetros / retorno de resultados
  - a2 – a7: seis registradores para a passagem de parâmetros.
  - ra: registrador de endereço de retorno.

- Instruções
  - jal ROTINA
    - Desvia a execução para o endereço indicado pelo label ROTINA e automaticamente salva o endereço de retorno (PC + 4) em ra.
  - jr ra
    - Desvia para a posição de memória armazenada em ra

RISC-V: Sub-rotina

- Convenção
  - Registradores "caller-saved": t0 – t6
    - Chamador é responsável por salvá-los em memória caso seja necessário usá-los novamente depois que um procedimento é chamado
  - Registradores "callee-saved": s2 - s11
    - Chamado é responsável por salvá-los em memória antes de utilizá-los e restaurá-los antes de devolver o controle ao chamador

C Code

```
void main()
{
  int y;
  y = sum(42, 7);
  ...
}

int sum(int a, int b)
{
  return (a + b);
}
```

RISC-V: Sub-rotina

```
        li a2, 10
        li a3, 21
        li a4, 31
        jal silly       # Now the result of the function is in a0.
        li a7, 10       # ecall code 10 is for exit.
        ecall           # exit the program gracefully
silly:
        add t0, a2, a3
        sub a0, a4, t0
        jr ra
```

Pilha

RISC-V: Pilha

- Preservação de Contexto
  - Quando as chamadas a procedimentos forem recursivas ou aninhadas, é conveniente o uso de uma pilha
  - Qualquer registrador usado pelo chamador deve ter seu conteúdo restaurado para o valor original antes da chamada
  - O conteúdo dos registradores é salvo na memória. Depois da execução do procedimento, estes registradores devem ter seus valores restaurados

RISC-V: Pilha

- Preservação de Contexto
  - sp (stack pointer) - registrador utilizado para guardar o endereço do topo da pilha:
    - a posição de memória que contêm os valores dos registradores salvos na memória pela última chamada
    - a posição a partir da qual a próxima chamada de procedimento pode salvar seus registradores

RISC-V: Pilha

- Manipulando a Pilha
  - RISC-V não possui instruções específicas para manipular a pilha
  - As instruções lw e sw devem ser utilizadas para essa função
    - Ex.: inserir o conteúdo do registrador $s0 na pilha

```
addi  sp, sp, -4   # avança o stack pointer
sw    s0, 0(sp)    # empilha o valor de s0
```

    - Ex.: remover o topo da pilha e armazenar em s0

```
lw    s0, 0(sp)   # restaura s0
addi  sp,sp,4     # desempilha o topo
```

Exceptions

RISC-V: Exceptions

- Exceção: condição incomum ocorrida em tempo de execução associada à uma instrução do programa em execução, como ao acessar um endereço de memória inválido ou executando uma instrução com um código de operação inválido.

- Interrupção: eventos externos assíncronos ao fluxo de instruções corrente, como um clique no botão do mouse.

- Trap: mecanismo de desvio de fluxo para lidar com exceções e interrupções, transferindo o controle para um manipulador (um trecho de código específico ou um tratamento de hardware customizado) de forma síncrona

RISC-V: Exceptions

- Cada nível de privilégio possui registrados para a manipulação de exceções denominados de registradores de controle e estado (control and status registers - CSRRs)
- OS CSRRs não fazem parte do banco de registradores
- Registradores no nível de máquina utilizados para a manipulação de exceções são:
  - mtvec: armazena o endereço do código de manipulação da exeção
  - mcause: armazena a causa da exceção
  - mepc (Exception PC): o PC onde a exceção ocorreu
  - mscratch: armazenamento temporário para o manipulador de exceção (stack pointers por exemplo)

RISC-V: Exceptions

- Instruções que acessam CSRRs são privilegiadas:

  csrr: CSR register read

  csrw: CSR register write

  csrrw: CSR register read/write

  mret: returns to address held in mepc

  Examplos:

```
csrr t1, mcause          # t1 = mcause
csrw mepc, t2            # mepc = t2
cwrrw t0, mscratch, t1   # t0 = mscratch
                         # mscratch = t1
```

RISC-V: Chamadas de Sistema

Interrupt/Exceção `mcause[XLEN-1]`	Código de Exceção `mcause[XLEN-2:0]`	Descrição
1	1	Interrupção de software de supervisor
1	3	Interrupção de software da máquina
1	5	Interrupção de temporizador de supervisor
1	7	Interrupção do temporizador da máquina
1	9	Interrupção externa do supervisor
1	11	Interrupção externa da máquina
0	0	Endereço de instrução desalinhado
0	1	Falha de acesso à instrução
0	2	Instrução ilegal
0	3	Ponto de parada
0	4	Endereço de load desalinhado
0	5	Falha de acesso de Load
0	6	Endereço do store desalinhado
0	7	Erro de acesso ao Store
0	8	Chamada de ambiente do U-mode
0	9	Chamada de ambiente do S-mode
0	11	Chamada de ambiente do M-mode
0	12	Falha na página de instruções
0	13	Falha na página de Load
0	15	Falha na página de Store

Fonte: https://biblioteca.univali.br/pergamumweb/vinculos/pdf/Guia%20pr%C3%A1tico%20RISC-V%20-%20atlas%20de%20uma%20arquitetura%20aberta.pdf

Chamadas de Sistema

RISC-V: Chamadas de Sistema

- System-call
  - Mecanismo que suspende a execução do programa do usuário e transfere o controle para o supervisor (S.O.) executado em um nível de privilégio maior
  - Concebido como uma exceção de chamada de sistema e implementado como uma instrução especial
    - Instrução ecall na ISA RISC-V
    - O supervisor acessa o conteúdo do registrador a7 para determinar qual tarefa está sendo solicitada e retorna o controle ao programa do usuário quando a tarefa é realizada
  - Possibilita chamar serviços fornecidos pelo supervisor como a leitura do teclado, a exibição de dados na tela, operações de acesso a disco, rede, etc..

RISC-V: Chamadas de Sistema

Tarefa	a7	Arguments / Result
Print Integer	1	a0 = integer value to print
Print Float	2	fa0 = float value to print
Print Double	3	fa0 = double value to print
Print String	4	a0 = address of null-terminated string
Read Integer	5	a0 = integer read
Read Float	6	fa0 = float read
Read Double	7	fa0 = double read
Read String	8	a0 = address of input buffer a1 = maximum number of characters to read
Exit	10	Exits the program with code 0
Print Char	11	a0 = character to print (only lowest byte is considered)
Read Char	12	a0 = character read

RISC-V: Chamadas de Sistema

1. Carregar argumentos das funções em registradores pré-definidos quando necessários
2. Carregar código da função no registrador a7
3. Executar a função ecall
4. Obter dados retornados quando disponíveis

a) li a0, 10 # argumento: a0=10
b) li a7, 1 # código da função "print integer"
c) ecall # exibe na tela conteúdo de a0 (10)

RISC-V: Chamadas de Sistema

- Finalizar a execução do programa

```
li a7, 10          # system call code 10 is for exit
ecall              # make the system call
```

- Leitura de inteiros

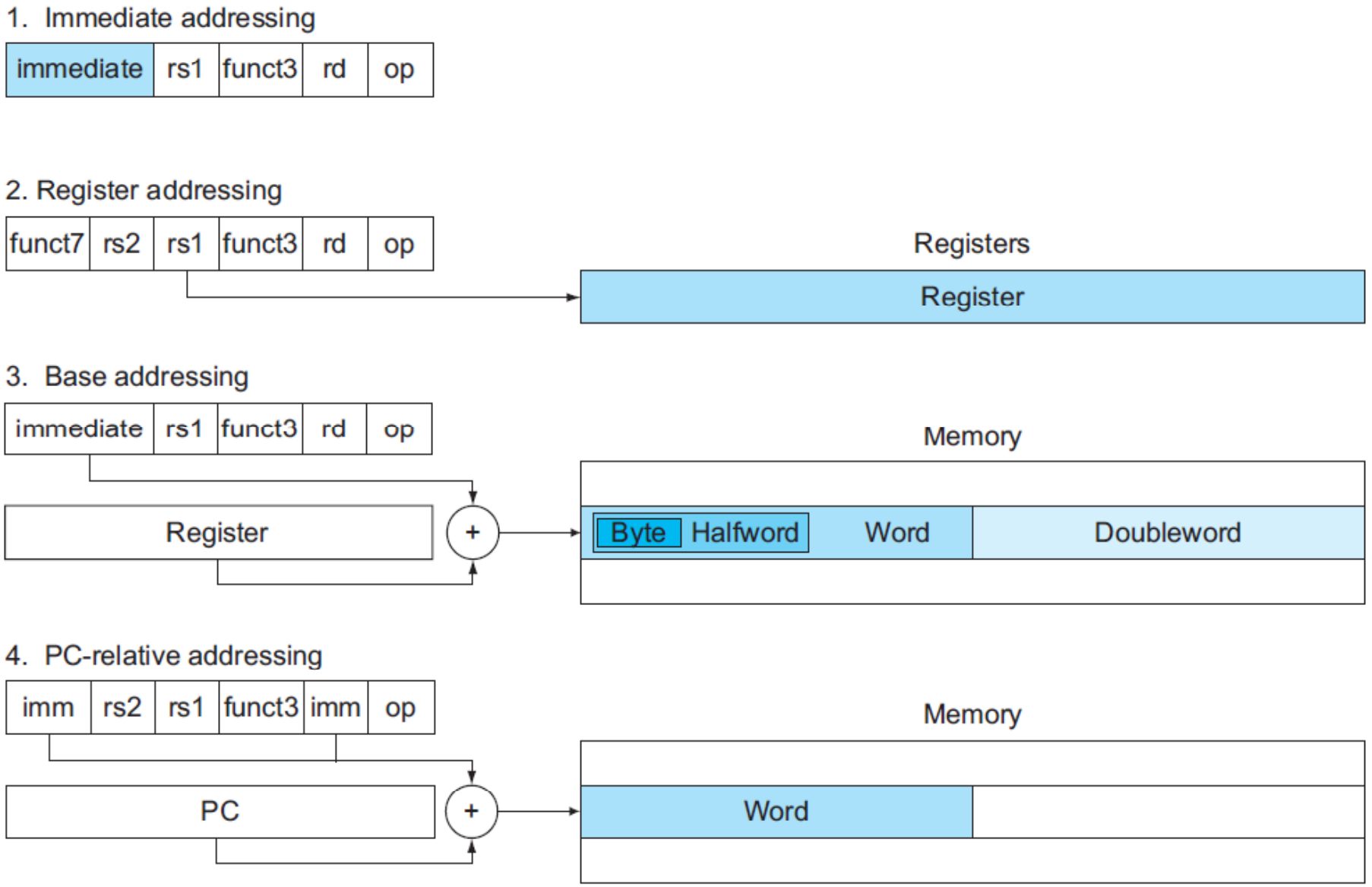
```
li   a7, 5         # load syscall read_int into a7
ecall              # make the system call
mv t0, a0          # move the number read into t0
```

- Impressão de inteiros

```
li  a0, 10         # argumento: a0 = 10
li  a7, 1          # código da função "print integer"
ecall              # exibe na tela conteúdo de a0 (10)
```

Modos de Endereçamentos

RISC-V: Modos de Endereçamento



RISC-V: Modos de Endereçamento

Endereçamento Imediato

- Campo imediato sinalisado de 12-bit utilizado como operando
  - `addi s4, t5, -73`
  - `ori  t3, t7, 0xFF`

Somente Registrador

- Operandos em registradores
  - `add s0, t2, t3`
  - `sub t6, s1, 0`

RISC-V: Modos de Endereçamento

Endereçamento com Base

- Loads and Stores
- Endereço do Operando:

  `base address + immediate`

  - `lw  s4, 72(zero)`
    - address = `0 + 72`

  - `sw  t2, -25(t1)`
    - address = `t1 - 25`

RISC-V: Modos de Endereçamento

Relativo ao PC: branches e **`jal`**

Address	**Instruction**
`0x354`	`L1: addi s1, s1, 1`
`0x358`	`sub t0, t1, s7`
`...`	`...`
`0xEB0`	`bne s8, s9, L1`

O label is (0xEB0-0x354) = 0xB5C (**2908**) bytes **anteriores à** bne

$imm_{12:0}$ = -2908	**1**		**0**	**1**	**0**	**0**		**1**	**0**	**1**	**0**		**0**	**1**	**0**	**0**
bit number	**12**		**11**	**10**	**9**	**8**		**7**	**6**	**5**	**4**		**3**	**2**	**1**	**0**

Assembly

`bne s8, s9, L1`
`(bne x24, x25, L1)`

Field Values

$imm_{12,10:5}$	rs2	rs1	funct3	$imm_{4:1,11}$	op
1100 101	24	25	1	**0010 0**	99
7 bits	5 bits	5 bits	3 bits	5 bits	7 bits

Machine Code

$imm_{12,10:5}$	rs2	rs1	funct3	$imm_{4:1,11}$	op	
1100 101	11000	11001	001	**0010 0**	110 0011	**(0xCB8C9263)**
7 bits	5 bits	5 bits	3 bits	5 bits	7 bits	