Laboratório de Sistemas
Embarcados e Distribuídos

Arquitetura e Programação do Processador RISC-V

Parte I

Histórico de revisões

Revisão	Data	Responsável	Descrição
0.1	03/2016	Prof. Cesar Zeferino	Primeira versão
0.2	03/2017	Prof. Cesar Zeferino	Detalhamento da arquitetura
0.3	08/2022	Prof. Felski	Mudança de arquitetura (RISC-V)

Introdução

- **Objetivo**
  - Conhecer o conceitos básicos sobre arquitetura e programação do processador RISC-V
- **Conteúdo**
  - O processador RISC-V
  - Arquitetura do RISC-V
  - Conjunto de instruções básico

Introdução

Bibliografia

- PATTERSON, David A.; HENNESSY, John L. Computer abstractions and technology. *In*: _____. **Computer organization and design:** the hardware/software interface: RISC-V Edition 1. ed. Amsterdan: Morgan Kauffman, 2018. cap. 2, p. 163~.

O processador RISC-V

- **RISC-V = Quinta Geração da Arquitetura RISC**
- **Desenvolvido pela University of California, Berkeley**
  - N25 e NX25 da Andes Technology Corporation, uma das primeiras empresas a apoiar a fundação RISC-V
  - Um SoC 64-bits de quatro núcleos e um SoC compatível com a plataforma Arduino criados pela SiFive
  - Uma família de processadores baseados nos subsets RV32GC e RV64GC produzidos pela Imperas para sistemas embarcados
  - GAP8, um processador 32-bits de 1+8 núcleos, além de um SoC e placa de desenvolvimento baseados em tal, criados pela GreenWaves Technologies

O processador RISC-V

- **O RISC-V é um processador do tipo RISC** (Reduced Instruction-set Processor)

- **Requisitos de processadores RISC**
  1. Executar uma operação em um único ciclo de relógio
  2. Todas as instruções com mesmo tamanho
  3. Acesso à memória apenas por instruções *load* e *store*
  4. Poucos formatos de instrução
  5. Poucas instruções
  6. Poucos modos de endereçamento
  7. Muitos registradores

O processador RISC-V

- HISTÓRIA
  - O projeto RISC-V iniciou-se quando Krste Asanović notou que haviam usos e viabilidade para um sistema computacional aberto, decidindo então desenvolver e publicar em um curto projeto de verão
  - O objetivo era criar um projeto com aplicações acadêmicas e industriais
  - O projeto também contou com o auxílio de David Patterson, um dos idealizadores da ideia original da arquitetura RISC e com financiamento inicial da DARPA
- 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

Arquitetura do RISC-V

Atributos arquiteturais RISC-V

Atributo	RISC-V
Tamanho da palavra de dados	32 / 64 bits
Tipos de dados	Inteiro e Real[pf]
Tamanho da palavra de instrução	32 bits
Formatos de instrução	4 (6)
Registradores de uso geral	32 (int) + 32 (pf)
Memória	2^{61} words

Arquitetura do RISC-V

Subconjunto básico de instruções

Classe da instrução	RISC-V
Aritmética	add, addi, sub
Memória	ld, lw, lh, sd, sw, sh
Desvio	beq, bne, blt, bge
Lógica	and, or, xor, andi, ori, xori
Procedimentos	jal, jalr
Deslocamento	sll, srl, sra, slli, srli, srai

- Observações
  - O quadro mostra um subconjunto das instruções dos dois processadores
  - O RISC-V consegue representar até 255 instruções e seus montadores suportam a programação com o uso de pseudo-instruções

Arquitetura do RISC-V

- **Operandos usados nas instruções**
  - Imediato
    - Constante de dado
    - Deslocamento de endereço
      - Na memória de dados: para acesso a variáveis
      - Na memória de instruções: para desvios
  - Registrador
    - Dado
    - Endereço base da memória de dados
    - Endereço da memória de instruções
  - O significado depende da classe da instrução

Arquitetura do RISC-V

- **Registradores de 32 bits / 64**
  - 32 registradores de inteiro: **`$0 - $31`**
  - 32 registradores de ponto flutuante: **`$f0 - $f31`**
  - Os registradores de inteiro têm nomes especiais em função do seu uso preferencial

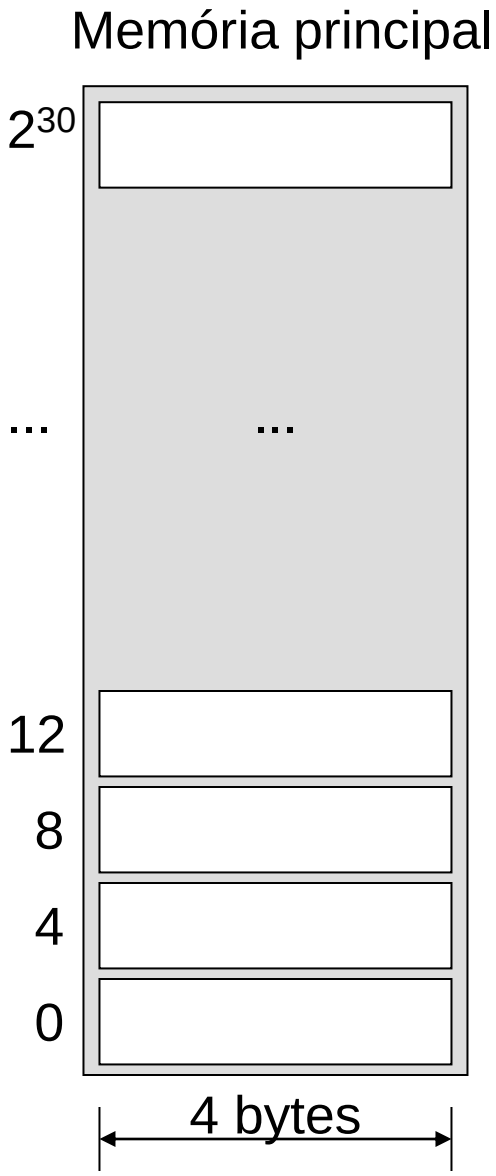
Reg	Nome	Reg	Nome	Reg	Nome	Reg	Nome
x0	zero	X8	$s0	X16	a6	X24	s8
X1	ra	X9	$s1	X17	a7	X25	s9
X2	sp	X10	a0	X18	s2	X26	s10
X3	gp	X11	a1	X19	s3	X27	s11
X4	tp	X12	a2	X20	s4	X28	t3
X5	t0	X13	a3	X21	s5	X29	t4
X6	t1	X14	a4	X22	s6	X30	t5
X7	t2	x15	a5	x23	s7	x31	t6

Arquitetura do RISC-V32i

Memória

- A memória do RISC-V32i é endereçada em nível de byte
- Palavras (words) de 32 bits consecutivas diferem de 4 unidades de endereço
- A memória principal do RISC-V32i pode ter até
  - 2^{32} Bytes = 4 Gbytes

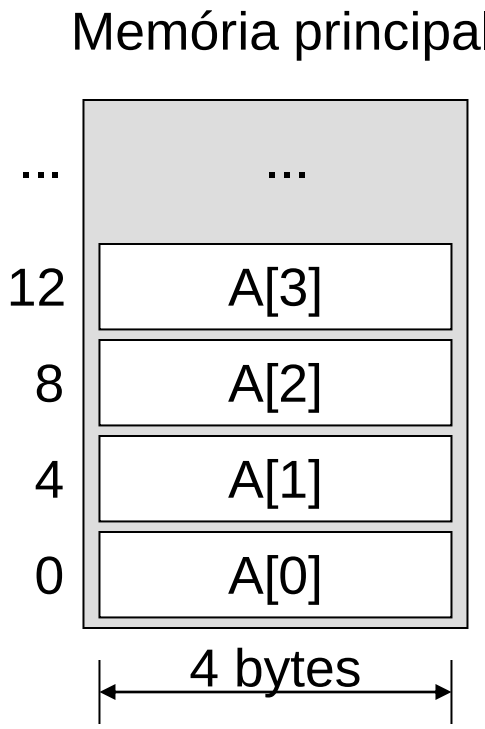
  ou
  - 2^{30} Words = 1 Gwords



Arquitetura do RISC-V32i

- **A memória principal pode ser vista como um grande array unidimensional (vetor) onde cada elemento do vetor ocupa uma palavra de 32 bits, ou seja, quatro posições de 1 byte**

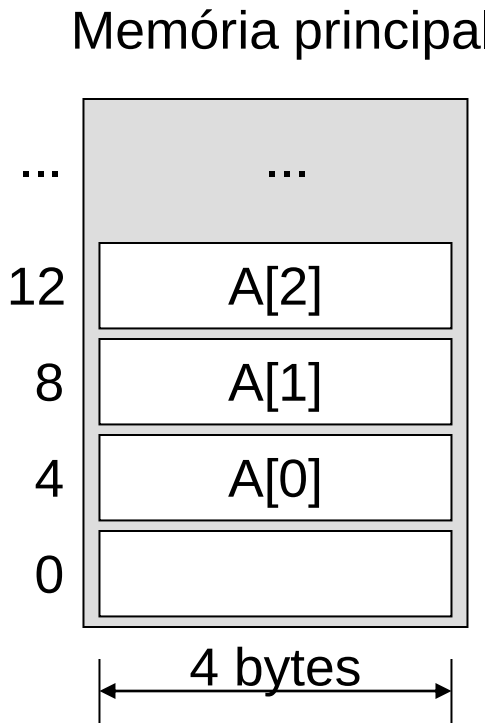
- **Se o elemento A[0] é armazenado na posição 0,**
  - O elemento A[1] está armazenado na posição **0+1x4 = 4**
  - O elemento A[2] está armazenado na posição **0+2x4 = 8**



Arquitetura do RISC-V

- **Se o elemento A[0] é armazenado na posição 4,**
  - O elemento A[1] está armazenado na posição **4+1x4 = 8**
  - O elemento A[2] está armazenado na posição **4+2x4 = 12**

A posição do elemento A[0] é considerada o endereço-base do vetor A

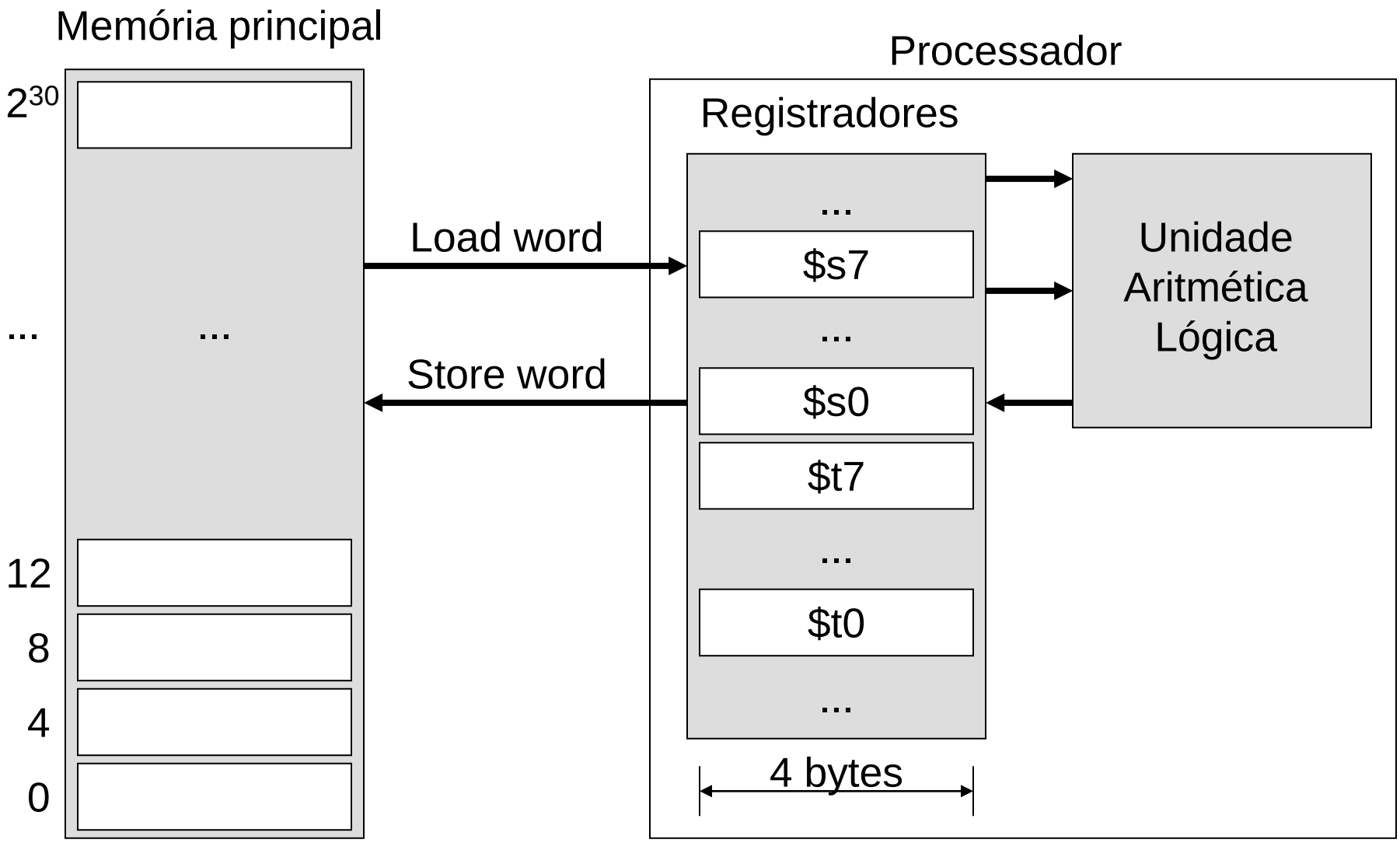


Arquitetura do RISC-V

- **O RISC-V é uma arquitetura RISC e nestas apenas as instruções de load e store podem acessar a memória**
- **Quando necessárias, em alguma operação aritmética, as variáveis armazenadas na memória devem ser carregadas (*load*) para um registrador**
- **Como o resultado da operação aritmética é mantido em um registrador, posteriormente, é necessário armazená-lo (*store*) em uma variável na memória principal**

Arquitetura do RISC-V

Visão simplificada do RISC-V



Formatos de instrução

Formato R (registrador)

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

Formato I (imediato)

Imediate	rs1	funct3	rd	opcode
12 bits	5 bits	3 bits	5 bits	7 bits

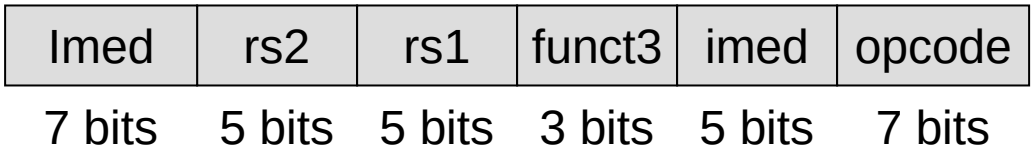
Formato S (source/*fonte*)

Imed	rs2	rs1	funct3	imed	opcode
7 bits	5 bits	5 bits	3 bits	5 bits	7 bits

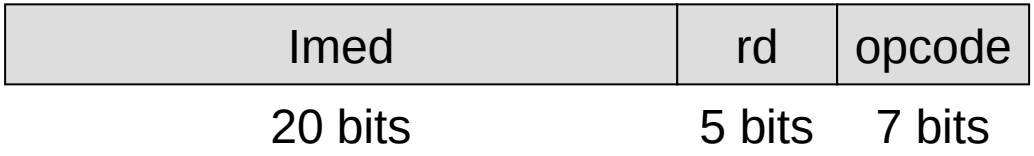
- O imediato é dividido em duas partes, bits 11 ao 5 na parte mais a esquerda e 4 ao 0 à direita

Formatos de instrução

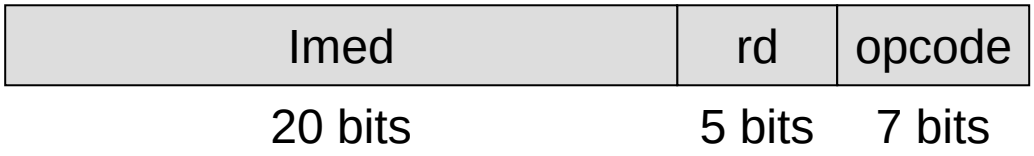
Formato SB



Formato U



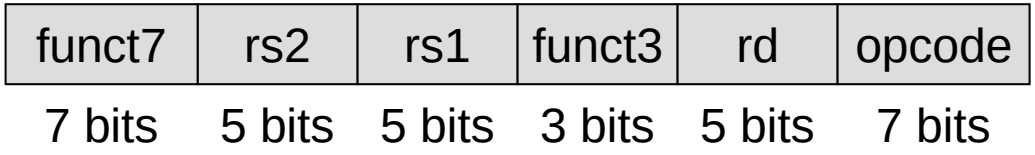
Formato UJ



Conjunto de instruções básico

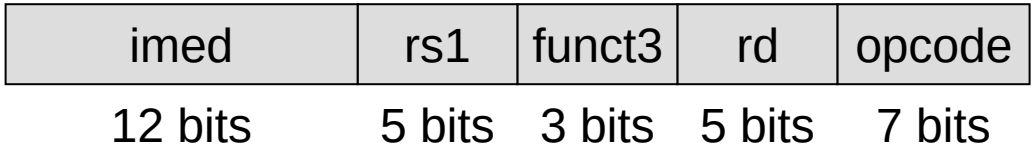
Classe da instrução	RISC-V
Aritmética	add, addi, sub
Memória	ld, lw, lh, sd, sw, sh
Desvio	beq, bne, blt, bge
Lógica	and, or, xor, andi, ori, xori, nor
Procedimentos	jal, jalr
Deslocamento	sll, srl, sra, slli, srli, srai

Conjunto de instruções básico: Formato R



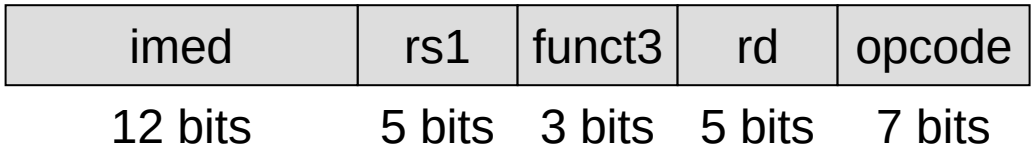

```
add   rd, rs1, rs2    # rd <- rs1 + rs2
sub   rd, rs1, rs2    # rd <- rs1 - rs2
sll   rd, rs1, rs2    # rd <- rs1 << rs2
slt   rd, rs1, rs2    # rd <- (rs1 < rs2)
xor   rd, rs1, rs2    # rd <- rs1 ^ rs2
srl   rd, rs1, rs2    # rd <- rs1 >> rs2
or    rd, rs1, rs2    # rd <- rs1 | rs2
and   rd, rs1, rs2    # rd <- ~(rs1 & rs2)
```

Conjunto de instruções básico: Formato I



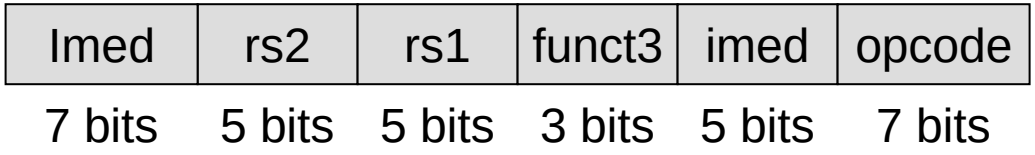

```
addi   rd, rs1, imed # rd <- rs1 + imed
slti   rd, rs1, imed # rd <- (rs1 < imed)
xori   rd, rs1, imed # rd <- rs1 ^ imed
ori    rd, rs1, imed # rd <- rs1 | imed
andi   rd, rs1, imed # rd <- rs1 & imed
slli   rd, rs1, imed # rd <- rs1 << imed
srli   rd, rs1, imed # rd <- rs1 >> imed
```

Conjunto de instruções básico: Formato I



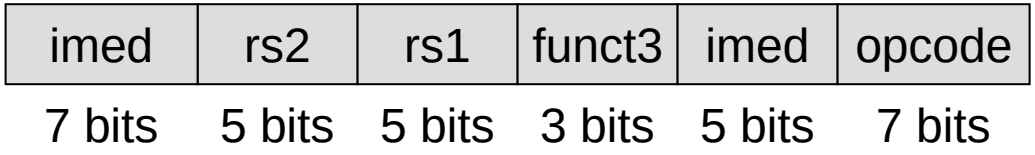

```
lb     rd, imed(rs1)  # rd <- Mem[imed + rs1]
lh     rd, imed(rs1)  # rd <- Mem[imed + rs1]
lw     rd, imed(rs1)  # rd <- Mem[imed + rs1]

jalr   rd, rs1, imed  # rd<-PC+4,
                        PC<-(PC+imed)&~1
```

Conjunto de instruções básico: Formato S



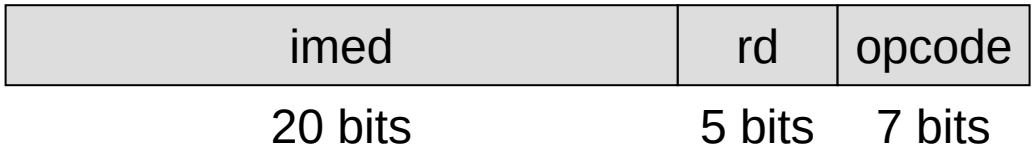

```
sb    rs2, imed(rs1)# Mem[imed + rs1] <- rs2
sh    rs2, imed(rs1)# Mem[imed + rs1] <- rs2
sw    rs2, imed(rs1)# Mem[imed + rs1] <- rs2
```

Conjunto de instruções básico: Formato B



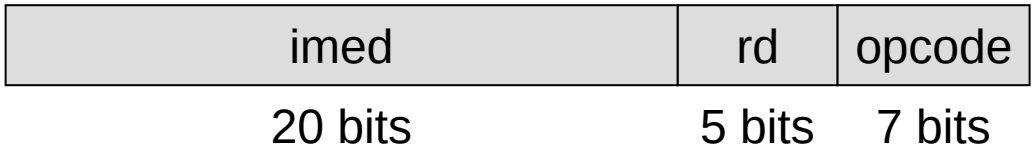

```
beq rs1, rs2, imed   # Se (rs1 = rs2) então
                     # PC <- PC+(imed<<2)

bne rs1, rs2, imed   # Se (rs1 != rs2) então
                     # PC <- PC+(imed<<2)

blt rs1, rs2, imed   # Se (rs1 < rs2) então
                     # PC <- PC+(imed<<2)

bge rs1, rs2, imed   # Se (rs1 >= rs2) então
                     # PC <- PC+(imed<<2)
```

Conjunto de instruções básico: Formato J




```
jal    rd, rs1, imed # rd<-PC+4,
                     PC<-(PC+imed)
```

Conjunto de instruções básico: Formato U




```
auipc rd, imed          # rd<-PC+imed << 12
lui   rd, imed          # rd<-imed << 12
```

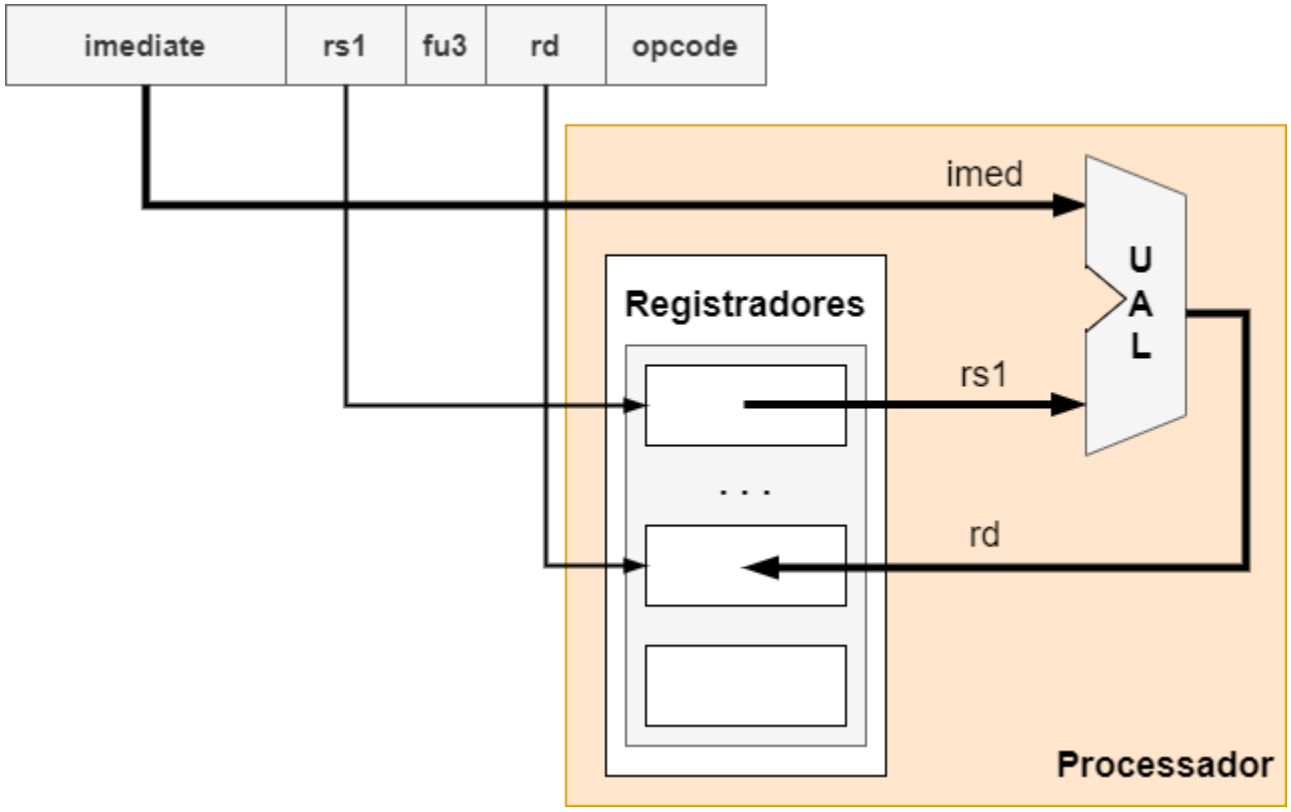
Modos de endereçamento

- **O modo de endereçamento especifica como é feito o acesso aos operandos da instrução**

- **No RISC-V, existem cinco modos de endereçamento**
  - Imediato
  - Via registrador
  - Via endereço-base
  - Relativo ao PC
  - Pseudo-direto

Modos de endereçamento

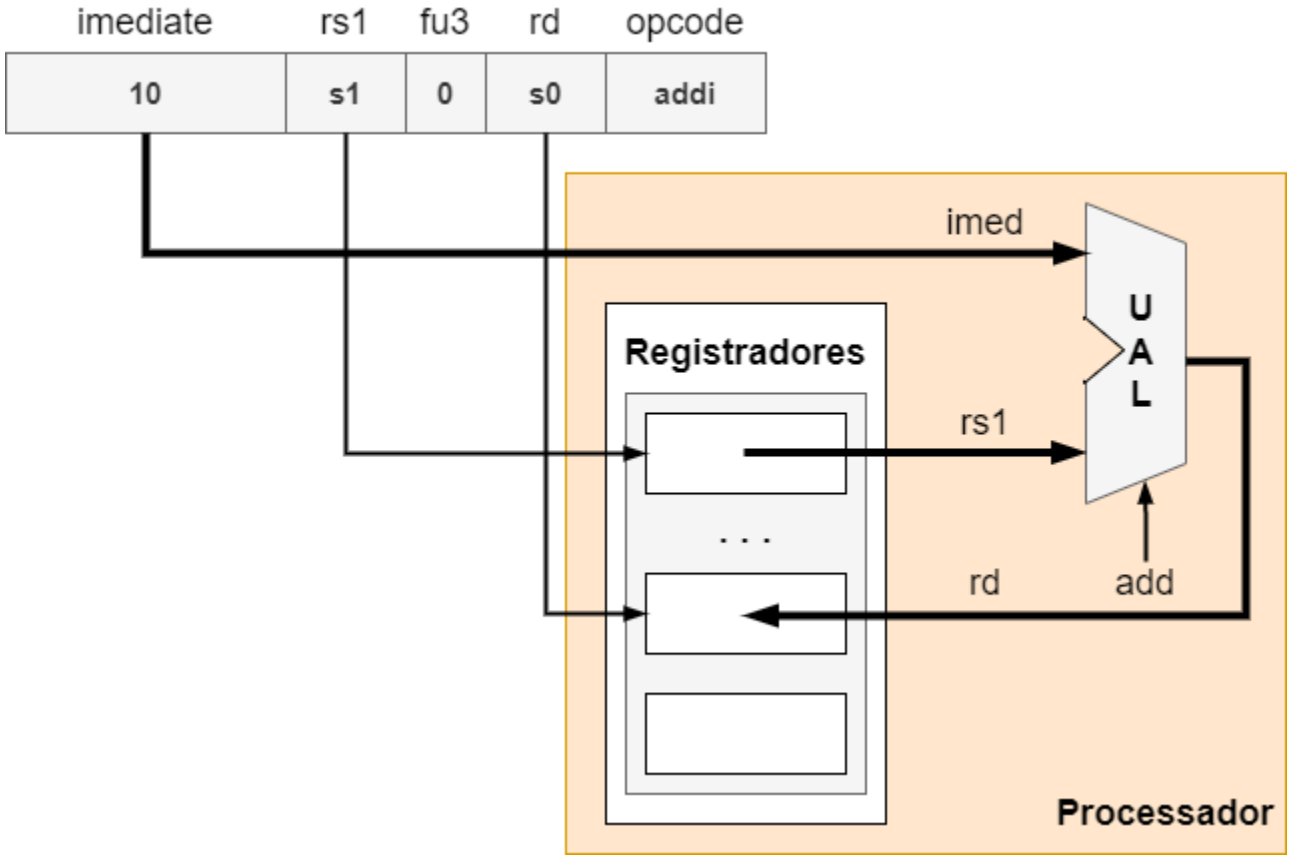
- **Endereçamento imediato**
  - Um dos operandos é uma constante (imediato) incluída na instrução e os outros dois são registradores (fonte e destino)
  - Usado na instrução `addi`



Modos de endereçamento

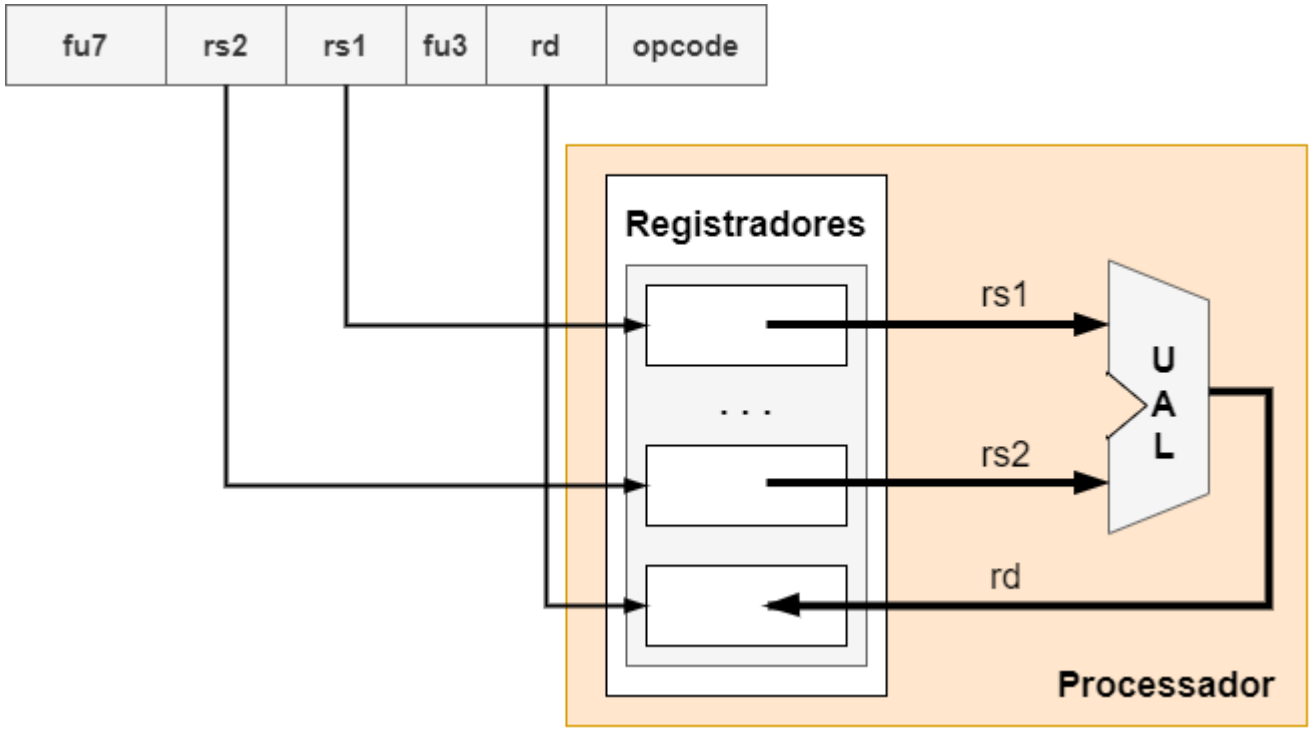
- **Endereçamento imediato – exemplo**

```
addi s0, s1, 10
```



Modos de endereçamento

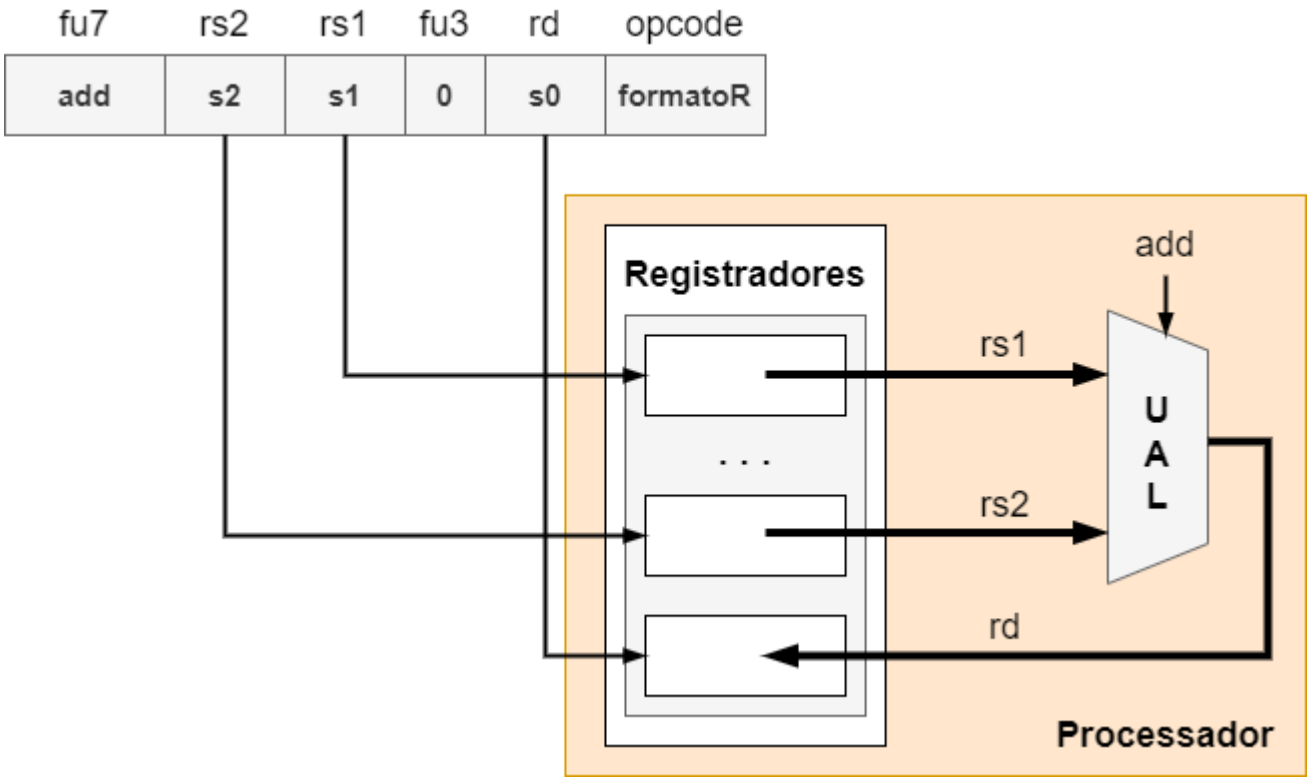
- **Endereçamento via registrador**
  - Todos os operandos da instrução estão armazenados em registradores são registradores apontados pelos campos *rs2*, *rs1* e *rd*
  - Usado nas instruções `add` e `sub`



Modos de endereçamento

- **Endereçamento via registrador – exemplo**

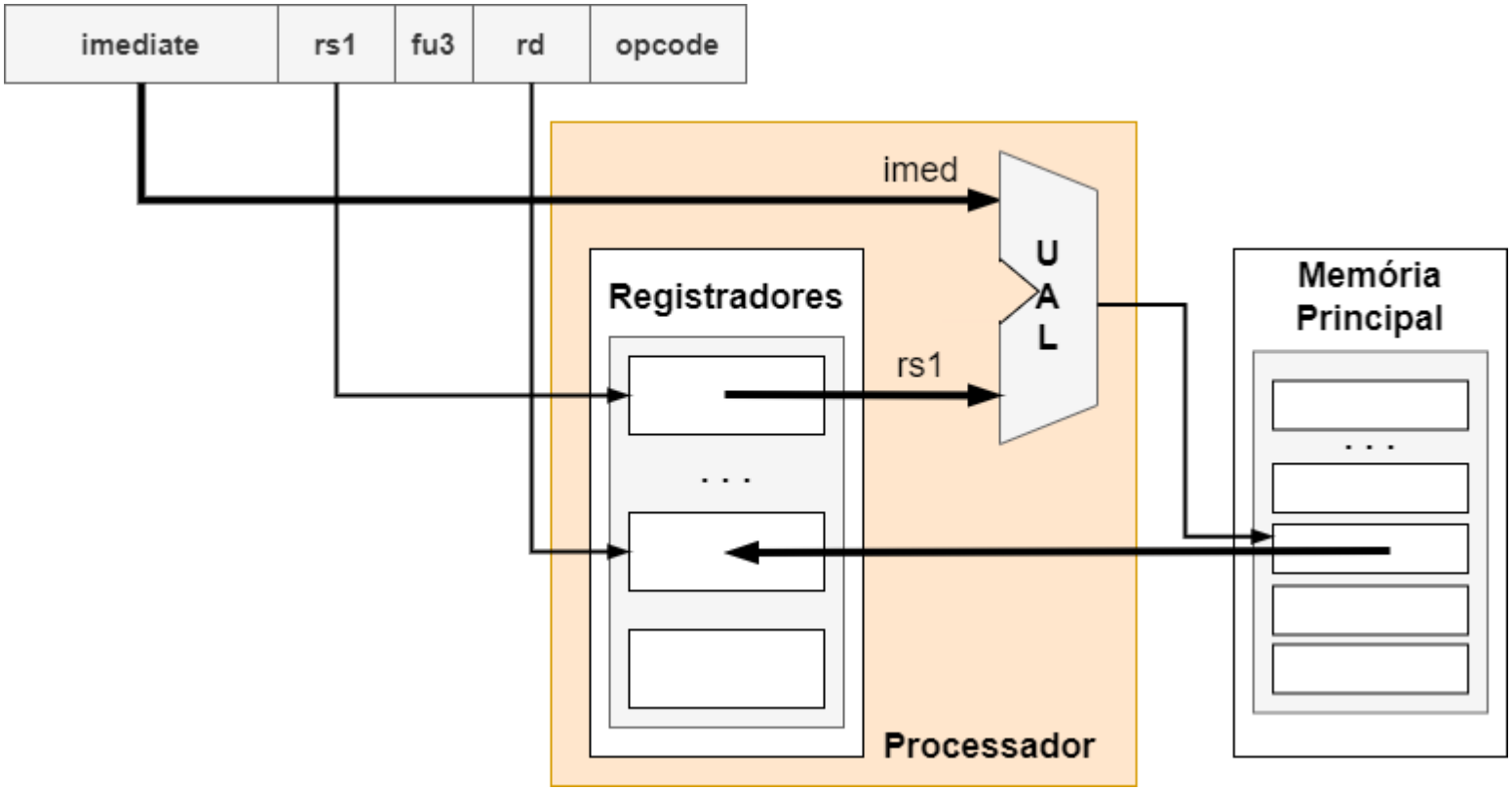
```
add s0, s1, s2
```



Modos de endereçamento

Endereçamento via registrador-base

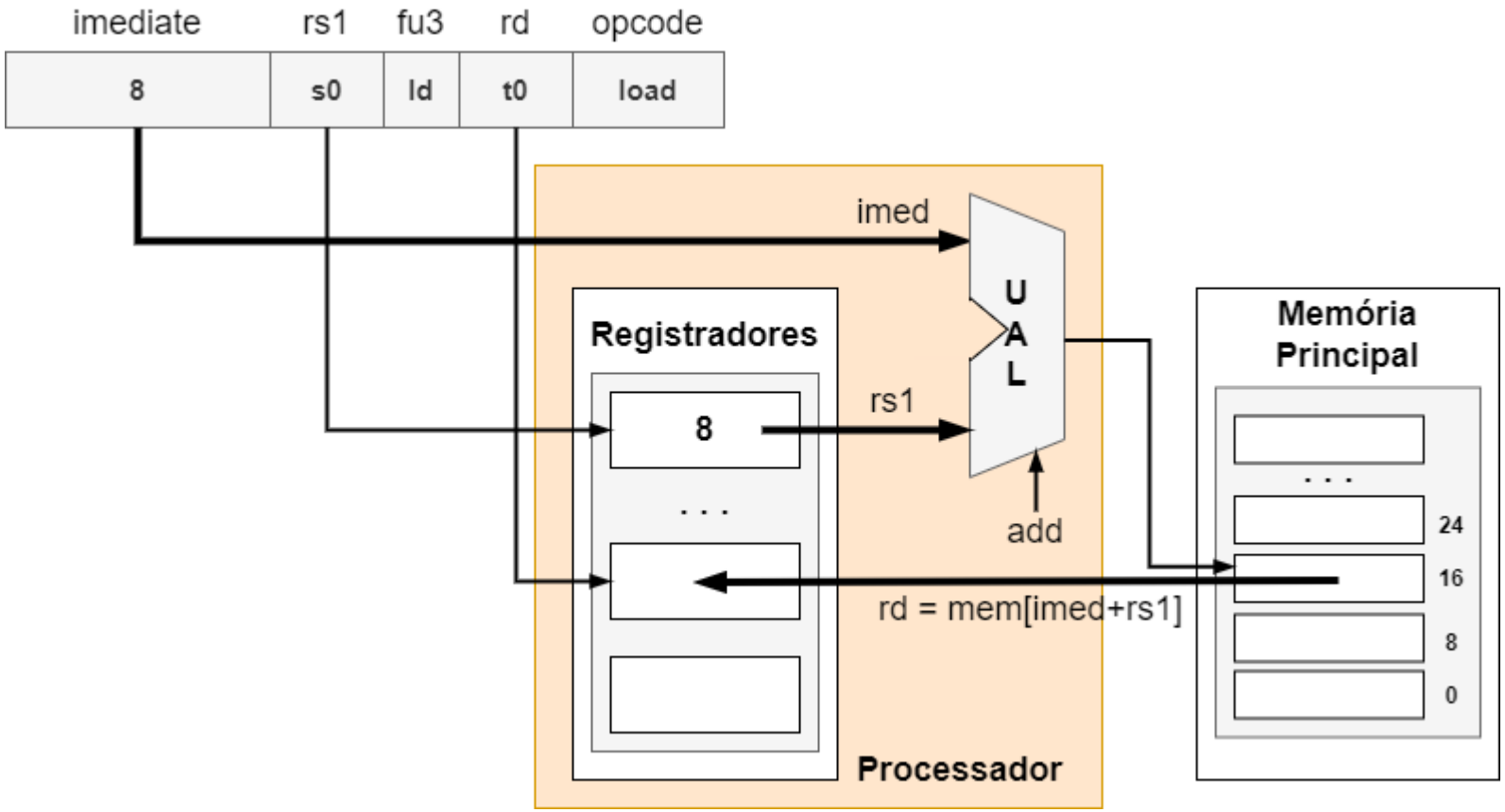
- Um dos operandos está armazenado na memória e seu endereço é indicado pela soma do campo imediato ao conteúdo de um registrador-base (campo rs)
- Usado nas instruções ld, lw, lh, lb



Modos de endereçamento

Endereçamento via registrador-base – exemplo 1

```
lw t0, 8(s0)
```

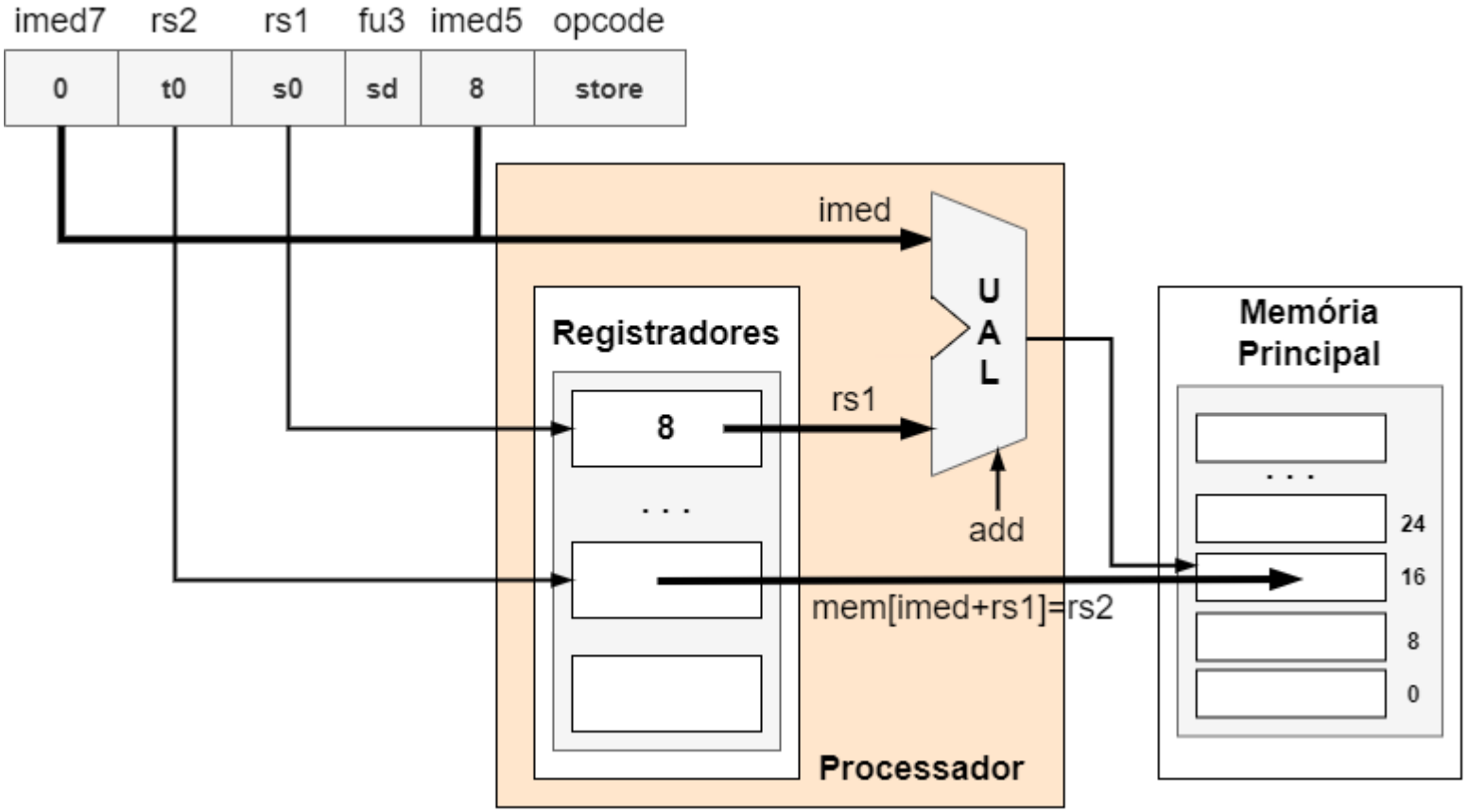


- Observe que o campo funct3 é usado para indicar o tipo de leitura
  - 0 = byte; 1 = half word; 2 = word; 3 = double word, 4 = byte unsigned, 5 = half word unsigned, 6 = word unsigned, 7 = double word unsigned

Modos de endereçamento

Endereçamento via registrador-base – exemplo 2

```
sw t0, 8(s0)
```

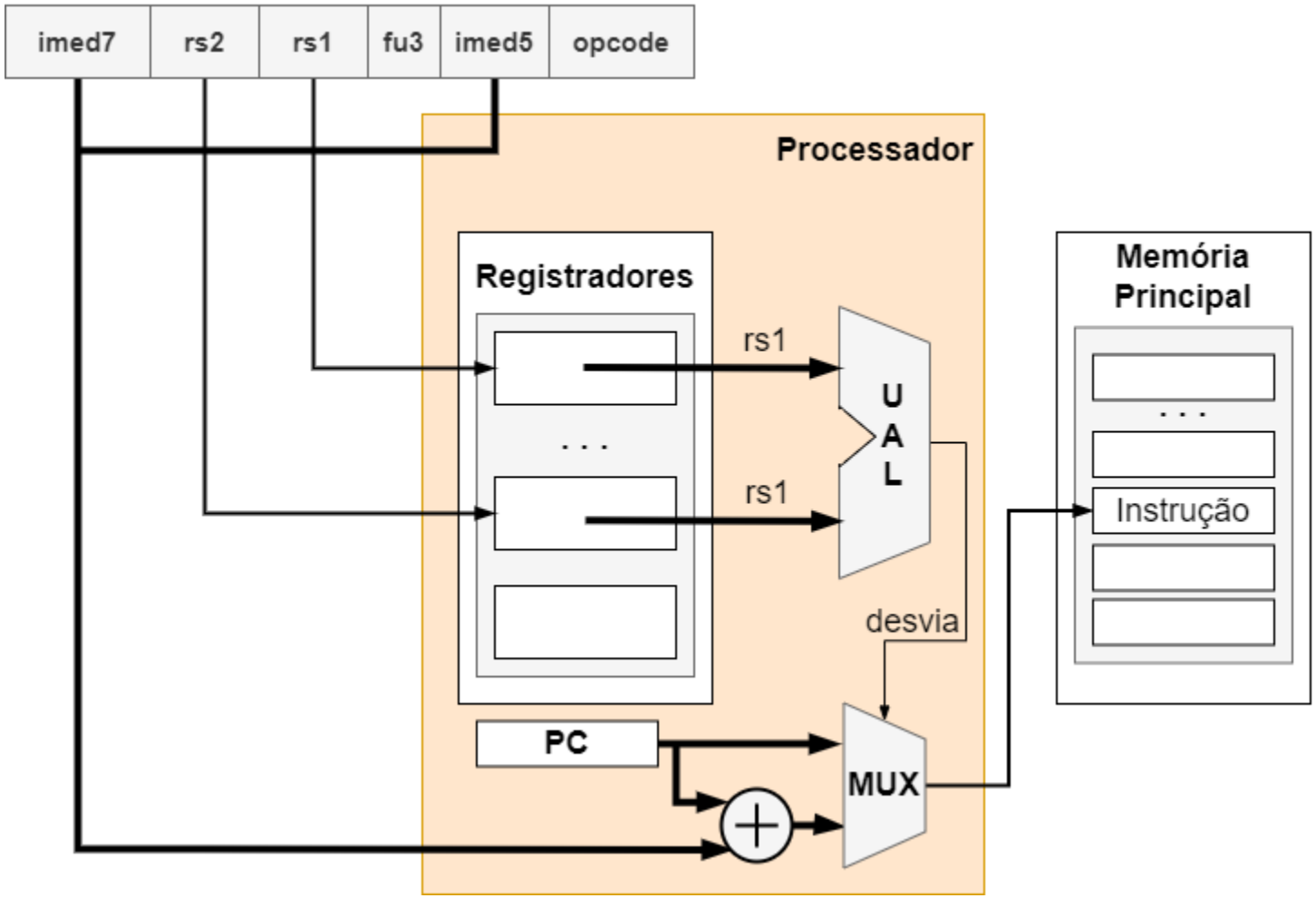


- Observe que o formato de instrução muda em relação ao lb
- Observe que o imediato é formado por imed7[11-5] e imed5[4-0]

Modos de endereçamento

Endereçamento relativo ao PC

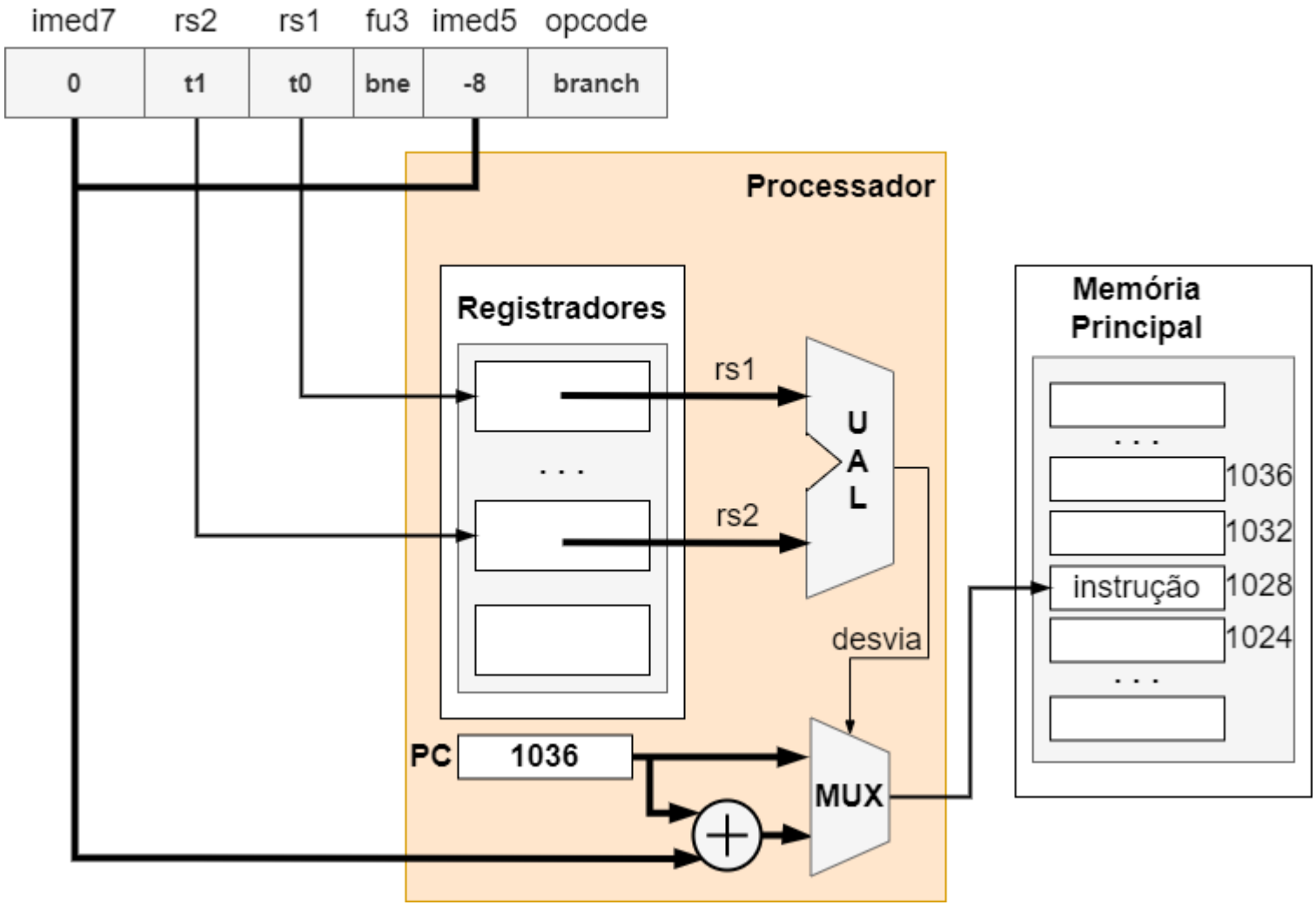
- O operando imediato (endereço-relativo) é somado ao PC (Program Counter)
  - beq
  - bne
  - blt
  - bge
  - bltu
  - bgeu
  - e



Modos de endereçamento

- **Endereçamento relativo ao PC – exemplo 1**

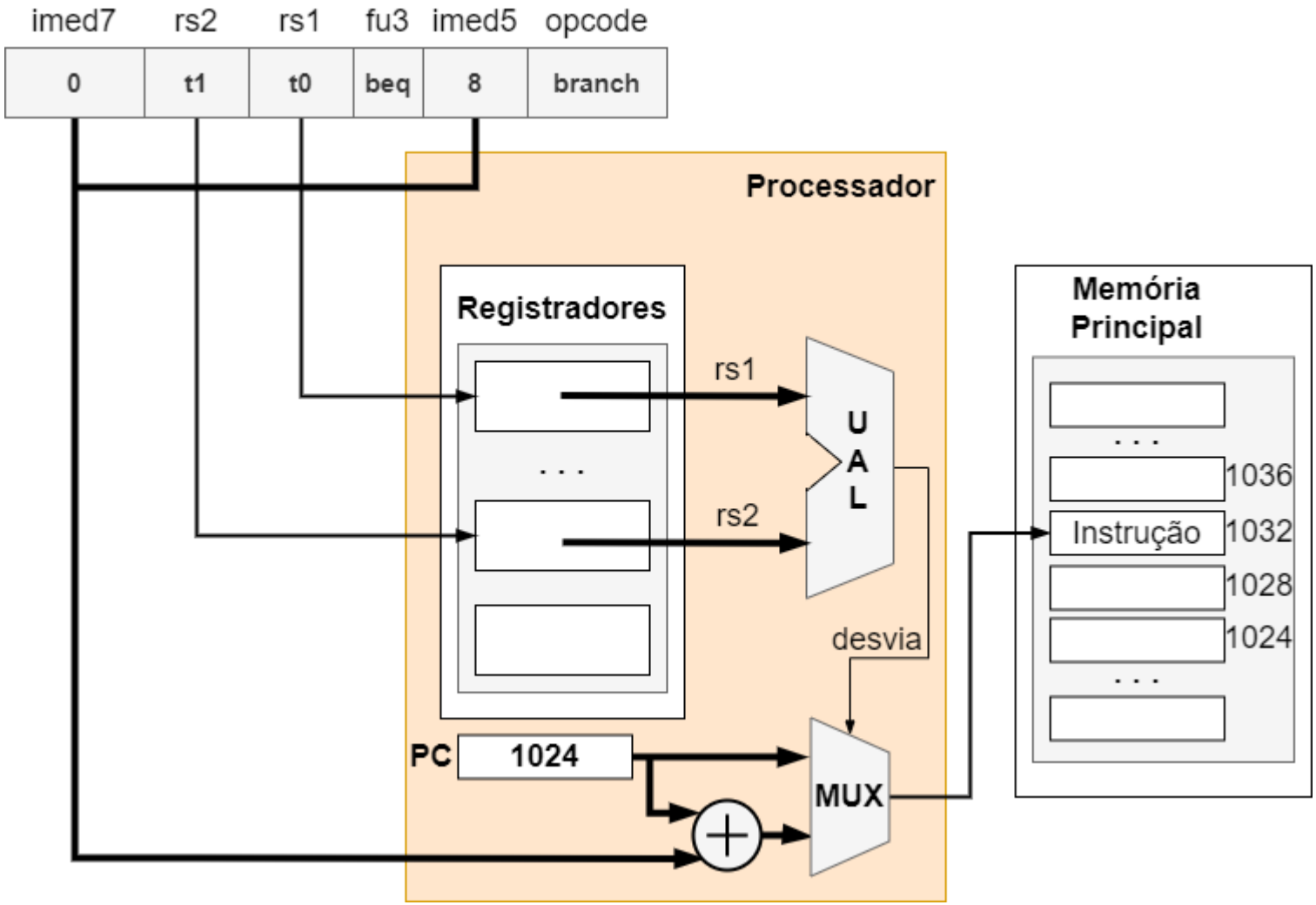
```
beq t0, t1, -8
```



Modos de endereçamento

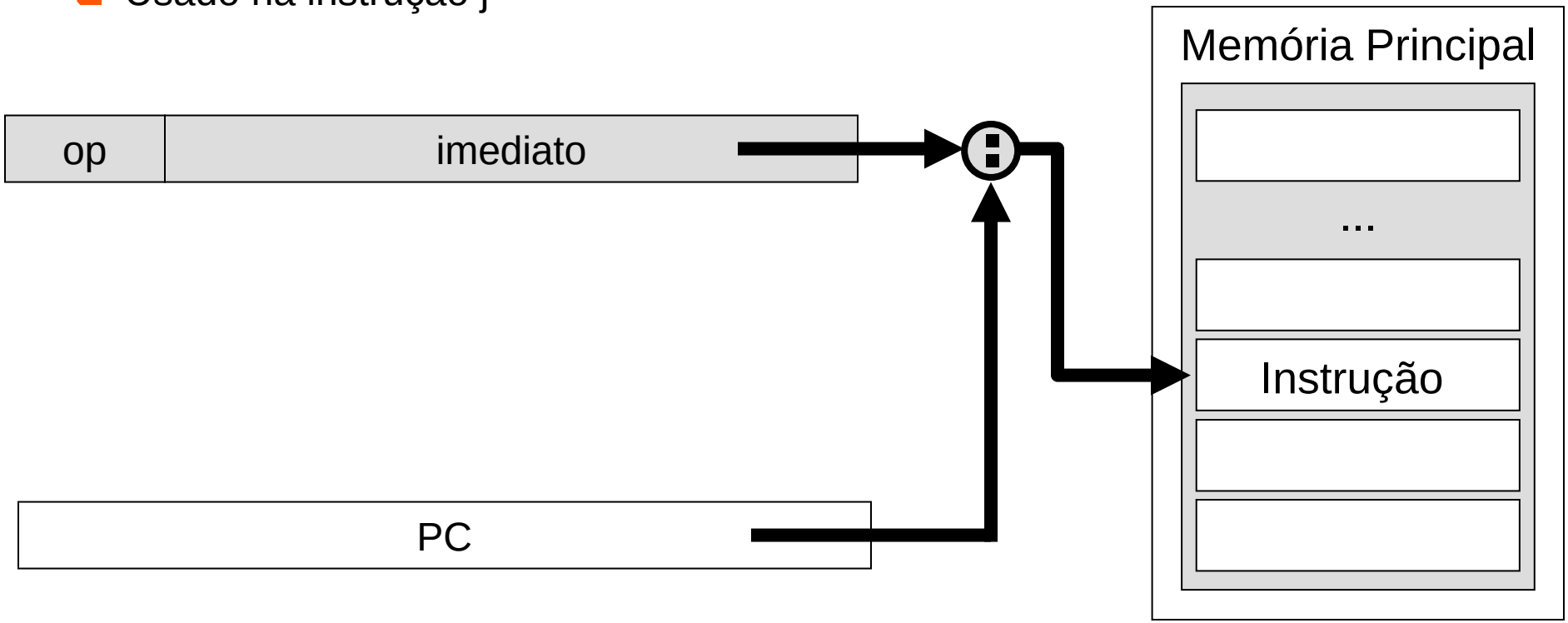
- **Endereçamento relativo ao PC – exemplo 2**

```
bne t0, t1, 8
```



Modos de endereçamento

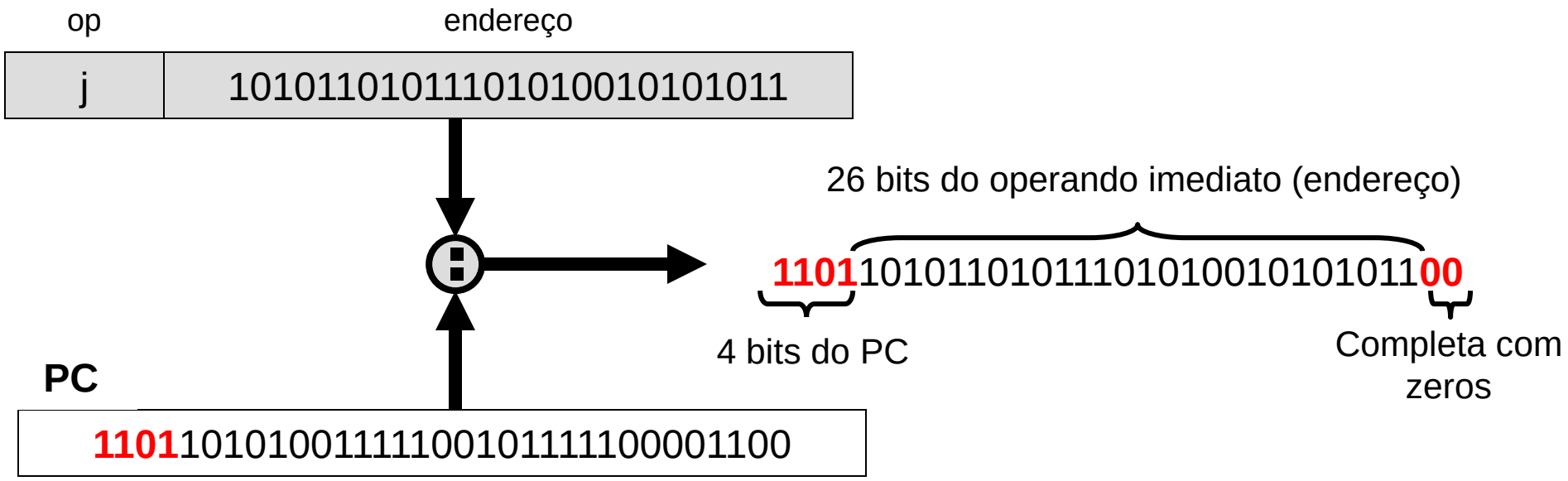
- **Endereçamento pseudodireto**
  - O operando imediato (endereço) é concatenado com os 4 bits mais significativos do PC
  - Usado na instrução j



Modos de endereçamento (REFAZER)

Endereçamento pseudodireto - exemplo

- O operando imediato (endereço) é concatenado com os 4 bits mais significativos do PC
- Usado na instrução j



CRIAR TABELA COM OS CÓDIGOS DE OPERAÇÃO

- TO DO