Organização e Arquitetura de Processadores

Assembly do MIPS

Formato de Instrução e Modo de Endereçamento

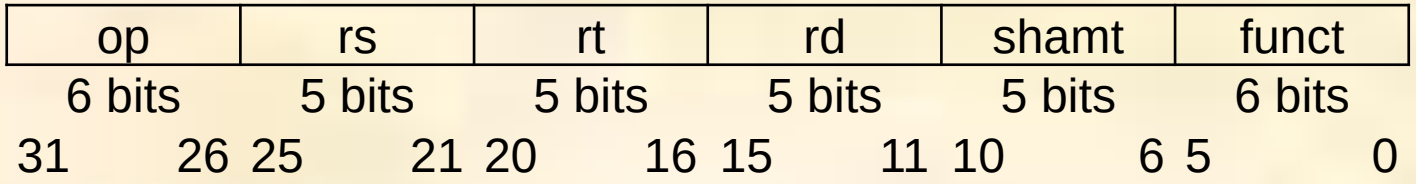
Principais ferramentas: ***MARS***
Apêndice A do livro do Patterson e Hennessy

Prof. Dr. César Marcon

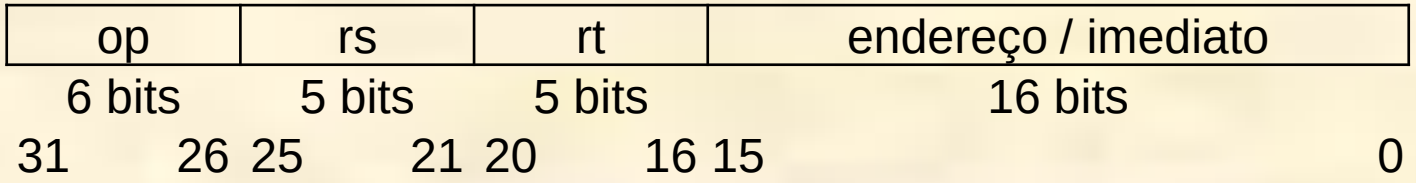
Formato de Instrução

- **O MIPS tem três formatos básicos de instrução**

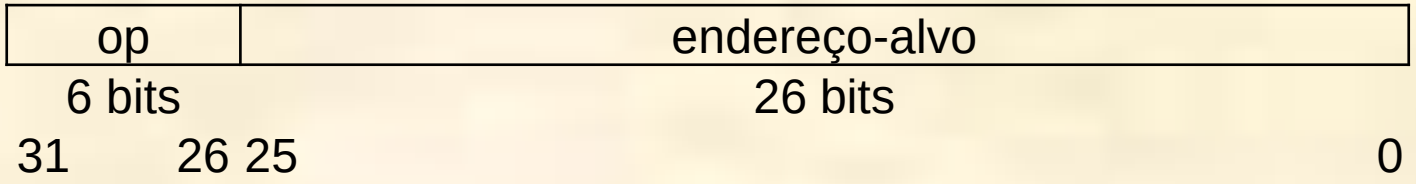
Instruções aritméticas



Transferência de dados e
Desvios condicionais



Salto incondicional

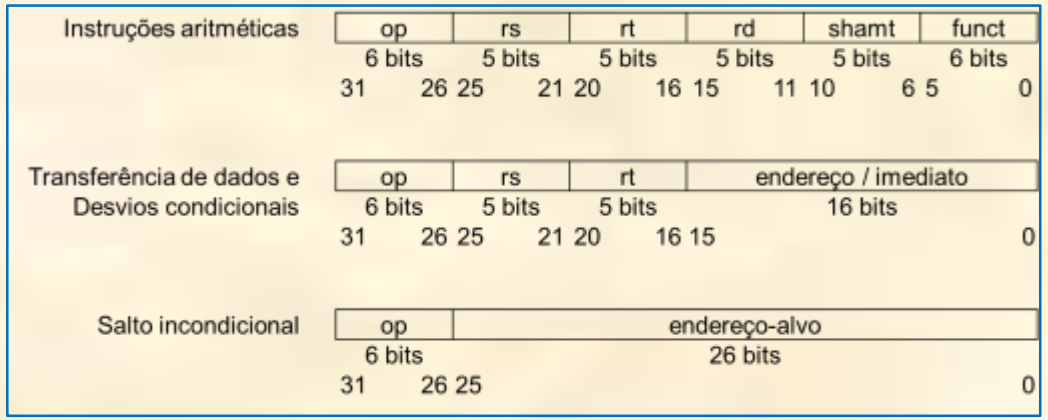


- **op** – código de operação**;** **funct –** código de função (associado ao campo op)
- **shamt –** campo para deslocamento; **rd** – registrador destino
- **rs** – registrador com primeiro operando fonte; **rt** – registrador que pode ser tanto o segundo operando fonte, como o destino, dependendo da instrução

Formato de Instrução – Análise das limitações

- **op com 6 bits → só permite $2^6 = 64$ instruções**
  - Aritméticas podem usar os campos shamt e funct para especializar a instrução
- **rs, rt, rd com 5 bits → banco de registradores limitado a 32**
- **endereço/imediato de 16 bits → somente consegue transferir dados dentro de um intervalo de ± 2^{15} bits (± 32.768)**
  - Endereçamento a word multiplica intervalo por 4 → ± 2^{17} bits (±131.072)
- **endereço-alvo de 26 bits → somente consegue fazer saltos de 2^{26} bits (67.108.864)**
  - Endereçamento a word multiplica intervalo por 4 → 2^{28} bits (268.435.456)

- **Limitações** restringem as possibilidades de descrições mais abstratas (mais fáceis de utilizar)
- **Pseudo-instrução** é uma solução, onde uma instrução de mais alto nível de abstração é transformada em outra equivalente ou quebrada em mais de uma instrução suportada pela arquitetura
- **Montador** converte pseudo-instrução em instruções da arquitetura alvo



Pseudo-Instrução – Exemplo com conversão do montador

```
int a = 4
int b = 7;
int maior;

main()
{
   if(a > b)
      maior = a;
   else
      maior = b;
}
```

```
.text
    .globl main
main:
    lw      $t0, a
    lw      $t1, b
    bgt     $t0, $t1, a_Maior
b_Maior:
    sw      $t1, maior
    j       fim
a_Maior:
    sw      $t0, maior
fim:
    li      $v0, 10
    syscall



.data
a: .word 4
b: .word 7
maior: .space 4
```

```
.text
    .globl main
main:
    lui     $at, a                  #
    lw      $t0, 0($at)             # lw   $t0, a
    lui     $at, b                  #
    lw      $t1, 4($at)             # lw   $t1, b
    slt     $at, $t1, $t0           #
    bne     $at, $zero, a_Maior # bgt  $t0, $t1, a_Maior
b_Maior:
    lui     $at, maior              #
    sw      $t1, 8($at)             # sw   $t1, maior
    j       fim                     # j    fim
a_Maior:
    lui     $at, maior              #
    sw      $t0, 8($at)             # sw   $t0, maior
fim:
    addiu   $v0, $zero, 10          # li   $v0, 10
    syscall                         # syscall
```

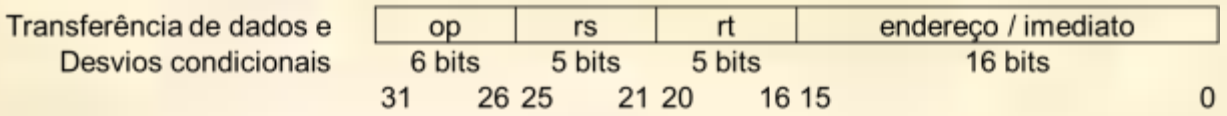
Pseudo-Instrução – Analisando Código Objeto

```
Address     Code        Basic                    Source
                                                 Line  Instruction                 Comment
0x00400000  0x3c011001  lui $1,0x00001001        16    lw $t0, a                   # t0 <- a
0x00400004  0x8c280000  lw $8,0x00000000($1)
0x00400008  0x3c011001  lui $1,0x00001001        17    lw $t1, b                   # t1 <- b
0x0040000c  0x8c290004  lw $9,0x00000004($1)
0x00400010  0x0128082a  slt $1,$9,$8             18    bgt   $t0, $t1, a_Maior     # if(a > b)
0x00400014  0x14200003  bne $1,$0,0x00000003
0x00400018  0x3c011001  lui $1,0x00001001        20    sw $t1, maior               #     maior = b;
0x0040001c  0xac290008  sw $9,0x00000008($1)
0x00400020  0x0810000b  j 0x0040002c             21    j  fim
0x00400024  0x3c011001  lui $1,0x00001001        23    sw $t0, maior               #     maior = a;
0x00400028  0xac280008  sw $8,0x00000008($1)
0x0040002c  0x2402000a  addiu $2,$0,0x0000000a   25    li $v0, 10
0x00400030  0x0000000c  syscall                  26    syscall                     # }
```

Pseudo-Instrução – Analisando exemplo

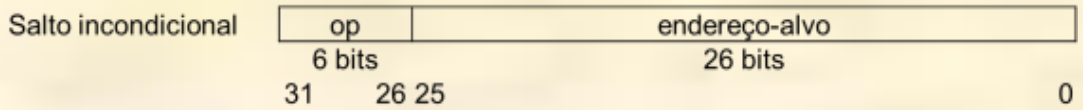
```
lui     $at, a_HIGH             #
lw      $t0, a_LOW($at)         # lw   $t0, a

lui     $at, maior_HIGH         #
sw      $t1, maior_LOW($at)     # sw   $t1, maior
```

- **a** e **maior** são endereços de 32 bits; endereço suporta apenas 16 bits
  - Requer pseudo-instrução



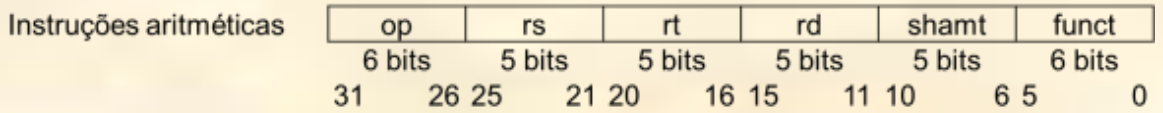

```
j       fim                     # j    fim
```

- **fim** é um deslocamento dentro de 2^{28} endereços
  - Não requer pseudo-instrução




```
syscall                         # syscall
```

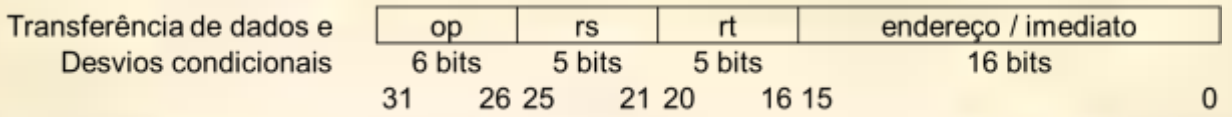
- **syscall** instrução compartilhada com aritméticas e lógicas
  - Usa o campo funct para diferenciar
  - Não requer pseudo-instrução



Pseudo-Instrução – Analisando exemplo

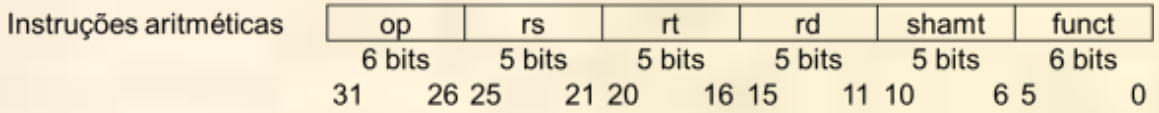
```
addiu  $v0, $zero, 10       # li   $v0, 10
```

- **imediato (10)** é um campo de 16 bits; instruções aritmética a registrador não têm espaço
  - Requer pseudo-instrução




```
slt    $at, $t1, $t0        #
bne    $at, $zero, a_Maior # bgt  $t0, $t1, a_Maior
```

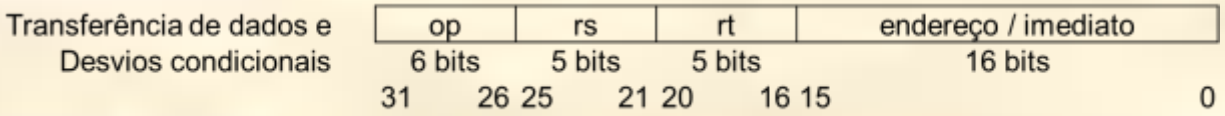
- **ULA** simplificada implementa teste de apenas o qualificador ZERO
  - Requer pseudo-instrução para implementar condições de desvios elaboradas (>, <, ≥, ≤)
  - Usa instrução aritmética **slt** à registrador para alterar **$at**



slt implementa >, <, ≥, ≤ possivelmente invertendo a pergunta!

```
bgt  $t0, $t1, a_Maior
```

- Usa instrução de desvio condicional para salto




```
slt  $at, $t1, $t0          # $at = $t1<$t0 ? 1 : 0
bne  $at, $zero, a_Maior    # se($t1!=0) PC ← a_Maior
```

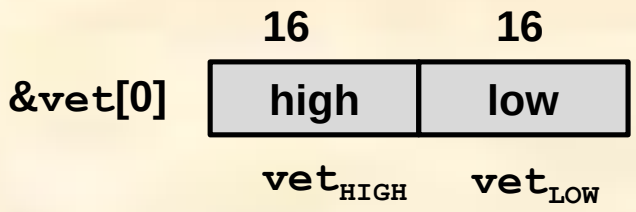
Outros Exemplos de Pseudo-Instrução

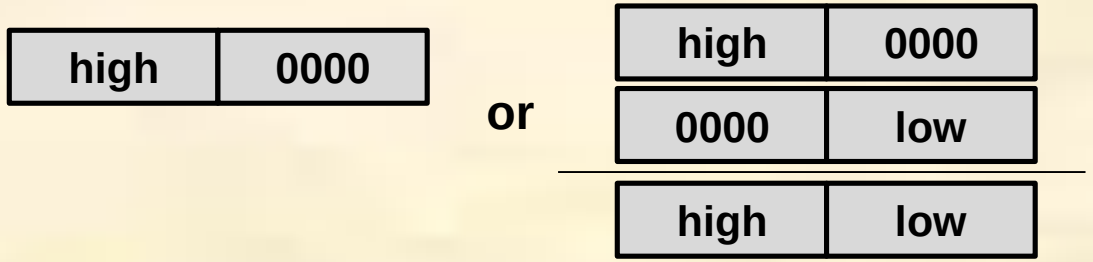
```
move   $t0, $zero
```

```
addu   $t0, $zero, $zero
```

```
la     $t0, vet
```

```
lui    $at, vet_HIGH          # parte alta do endereço
ori    $t0, $at, vet_LOW      # parte baixa do endereço
```






```
beqz   $t0, salto
```

```
beq    $t0, $zero, salto
```

```
subi   $t0, $t0, 1
```

```
addi   $at, $zero, 1
sub    $t0, $t0, $at
```

- **Atenção!** Saber pseudo e instruções pode reduzir área a aumentar a velocidade de operação

```
subi   $sp, $sp, 8
```

```
addi   $sp, $sp, -8
```

```
addi   $at, $zero, 8
sub    $sp, $sp, $at
```

```
addi   $sp, $sp, -8
```

Codificando uma Instrução

lw rt, Offset(rs)

0x23	rs	rt	Offset
6	5	5	16

0x23	$t2	$t0	-12

⇐ `lw $t0, -12($t2)`

```
Código   RS    RT        Offset
100011 01010 01000 1111111111110100  ➔
```

```
Em Binário (agrupados em 4 bits)
1000 1101 0100 1000 1111 1111 1111 0100
Em Hexadecimal (conforme agrupamento)
   8    D    4    8    F    F    F    4
```

```
100011: 0010 0011
          2    3 -> 0x23 = 35

01010:  0000 1010
          0    A -> 0x0A = 10   ➔ $t2

01000:  0000 1000
          0    8 -> 0x08 = 8    ➔ $t0

1111111111110100: 1111 1111 1111 0100
                    F    F    F    4 -> 0xFFF4
                                      -> 0xFFFFFFF4 = -12 (extensão de sinal)
```

Número	Nome	Significado
0	$zero	constante 0
1	$at	reservado para o assembly
2, 3	$v0, $v1	resultado de função
4 - 7	$a0 - $a3	argumento para função
8 - 15	$t0 - $t7	temporário
16 - 23	$s0 - $s7	temporário (salvo nas chamadas de função)
24, 25	$t8, $t9	temporário
26, 27	$k0, $k1	reservado para o SO
28	$gp	apontador de área global
29	$sp	stack pointer
30	$fp	frame pointer
31	$ra	registrador de endereço de retorno

Pseudo-Instrução – Analisando Código Objeto

Address	Code	Basic	Source Line	Instruction	Comment
0x00400000	0x3c011001	lui $1,0x00001001	16	lw $t0, a	# t0 <- a
0x00400004	0x8c280000	lw $8,0x00000000($1)			
0x00400008	0x3c011001	lui $1,0x00001001	17	lw $t1, b	# t1 <- b
0x0040000c	0x8c290004	lw $9,0x00000004($1)			
0x00400010	0x0128082a	slt $1,$9,$8	18	bgt $t0, $t1, a_Maior	# if(a > b)
0x00400014	0x14200003	bne $1,$0,0x00000003	*3*4 + 0x400018 = 0x400024*		
0x00400018	0x3c011001	lui $1,0x00001001	20	sw $t1, maior	# maior = b;
0x0040001c	0xac290008	sw $9,0x00000008($1)			
0x00400020	0x0810000b	j 0x0040002c	21	j fim	
0x00400024	0x3c011001	lui $1,0x00001001	23	sw $t0, maior	# maior = a;
0x00400028	0xac280008	sw $8,0x00000008($1)			
0x0040002c	0x2402000a	addiu $2,$0,0x0000000a	25	li $v0, 10	
0x00400030	0x0000000c	syscall	26	syscall	# }
0x10010000	0x00000004				# a: .word 4
0x10010004	0x00000007				# b: .word 7
0x10010008	0x00000000				# maior: .space 4

Analisando Instruções (1)

Transferência de dados e Desvios condicionais

op	rs	rt	endereço / imediato
6 bits	5 bits	5 bits	16 bits

31 26 25 21 20 16 15 0

Salto incondicional

op	endereço-alvo
6 bits	26 bits

31 26 25 0

Instruções aritméticas

op	rs	rt	rd	shamt	funct
6 bits	5 bits	5 bits	5 bits	5 bits	6 bits

31 26 25 21 20 16 15 11 10 6 5 0

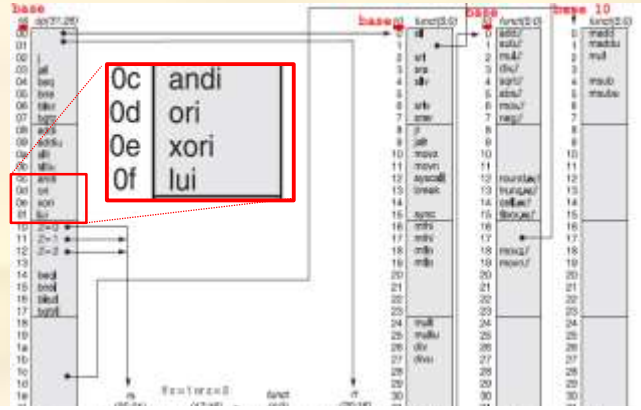
0x00400000 0x3c011001 lui $1,0x00001001

- Codificação hexadecimal: 0x3c011001
- Codificação binária: 0011 1100 0000 0001 0001 0000 0000 0001
- Descobrindo o código: [001111] → 0x0F → lui

Load superior imediato

lui rt, imm

0xf	0	rt	imm
6	5	5	16



- Reagrupando: [001111][00000][00001][0001 0000 0000 0001]
- Agrupando em hexa: 0xF 0x0 0x1 0x1001
- Identificando o assembly: lui $at, 0x1001

Número	Nome	Significado
0	$zero	constante 0
1	$at	reservado para o assembly
2, 3	$v0, $v1	resultado de função
4 - 7	$a0 - $a3	argumento para função
8 - 15	$t0 - $t7	temporário
16 - 23	$s0 - $s7	temporário (salvo nas chamadas de função)
24, 25	$t8, $t9	temporário
26, 27	$k0, $k1	reservado para o SO
28	$gp	apontador de área global
29	$sp	stack pointer
30	$fp	frame pointer
31	$ra	registrador de endereço de retorno

Analisando Instruções (2)

Transferência de dados e Desvios condicionais

op	rs	rt	endereço / imediato
6 bits	5 bits	5 bits	16 bits

31 26 25 21 20 16 15 0

Salto incondicional

op	endereço-alvo
6 bits	26 bits

31 26 25 0

Instruções aritméticas

op	rs	rt	rd	shamt	funct
6 bits	5 bits	5 bits	5 bits	5 bits	6 bits

31 26 25 21 20 16 15 11 10 6 5 0

```
0x00400004    0x8c280000    lw $8,0x00000000($1)
```

- Codificação hexadecimal: 0x8c280000
- Codificação binária: 1000 1100 0010 1000 0000 0000 0000 0000
- Descobrindo o código: [100011] → 0x23 → lw

Load word

lw rt, Offset(rs)

0x23	rs	rt	Offset
6	5	5	16

- Reagrupando: [100011][00001][01000][0000 0000 0000 0000]
- Agrupando em hexa: 0x23 0x1 0x8 0x0000
- Identificando o assembly: lw $t0, 0($at)

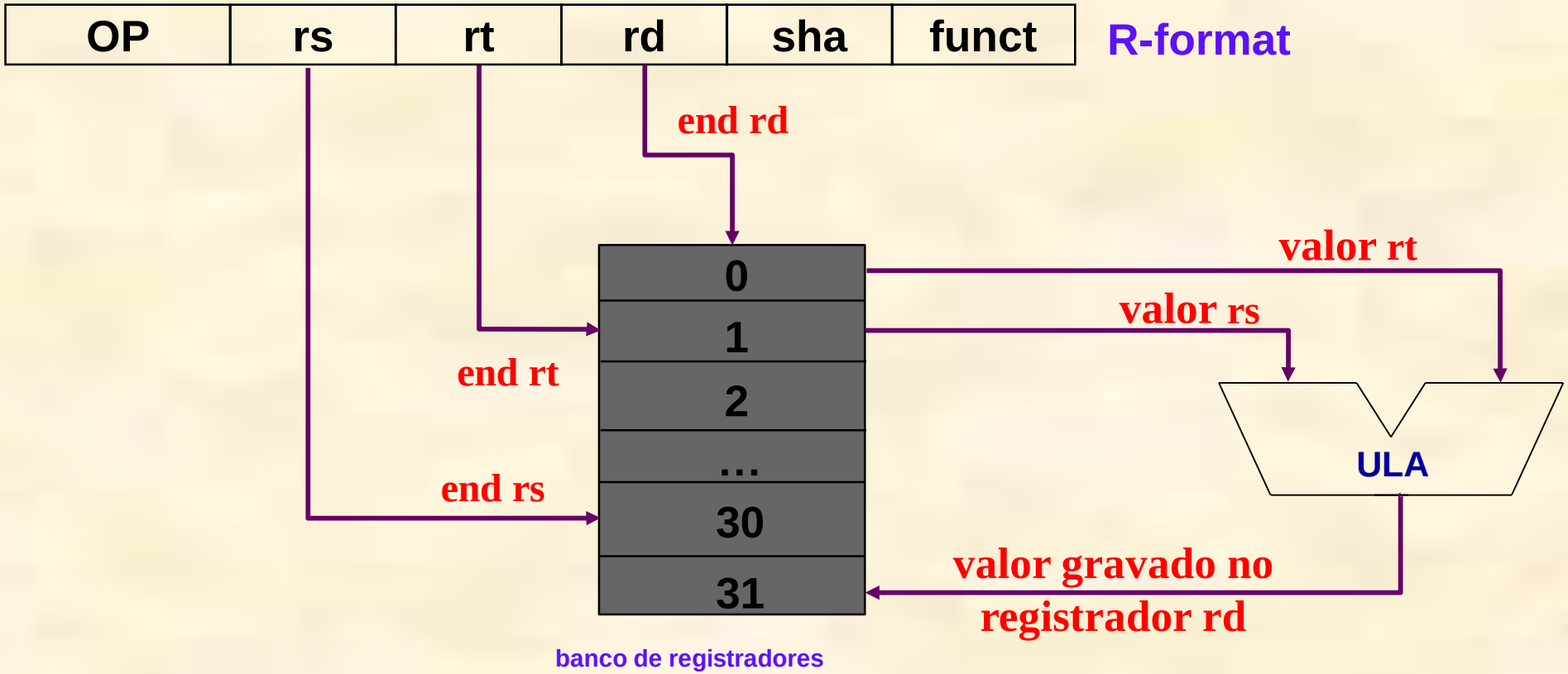
Número	Nome	Significado
0	$zero	constante 0
1	$at	reservado para o assembly
2, 3	$v0, $v1	resultado de função
4 - 7	$a0 - $a3	argumento para função
8 - 15	$t0 - $t7	temporário
16 - 23	$s0 - $s7	temporário (salvo nas chamadas de função)
24, 25	$t8, $t9	temporário
26, 27	$k0, $k1	reservado para o SO
28	$gp	apontador de área global
29	$sp	stack pointer
30	$fp	frame pointer
31	$ra	registrador de endereço de retorno

Modo de Endereçamento

- **O modo de endereçamento define a forma que o processador emprega para acessar dados**
  - Define o formato de instrução
  - Depende da hierarquia da memória de dados (registrador, ...)
- **Máquinas RISC**
  - Acessam a memória através de instruções load/store
  - Tem poucos modos de endereçamento
  - Usam menos modos de endereçamento com memória e mais com registrador
- **Máquinas CISC**
  - Podem trabalhar com leitura e escrita simultânea na memória
  - Empregam mais modos de endereçamento e com vários modelos
- **MIPS é uma máquina RISC com cinco modos de endereçamento**
  - À registrador
  - Imediato
  - Base-Deslocamento
  - Relativo ao PC
  - Direto à (Registrador/Imediato)

Modo de Endereçamento à Registrador

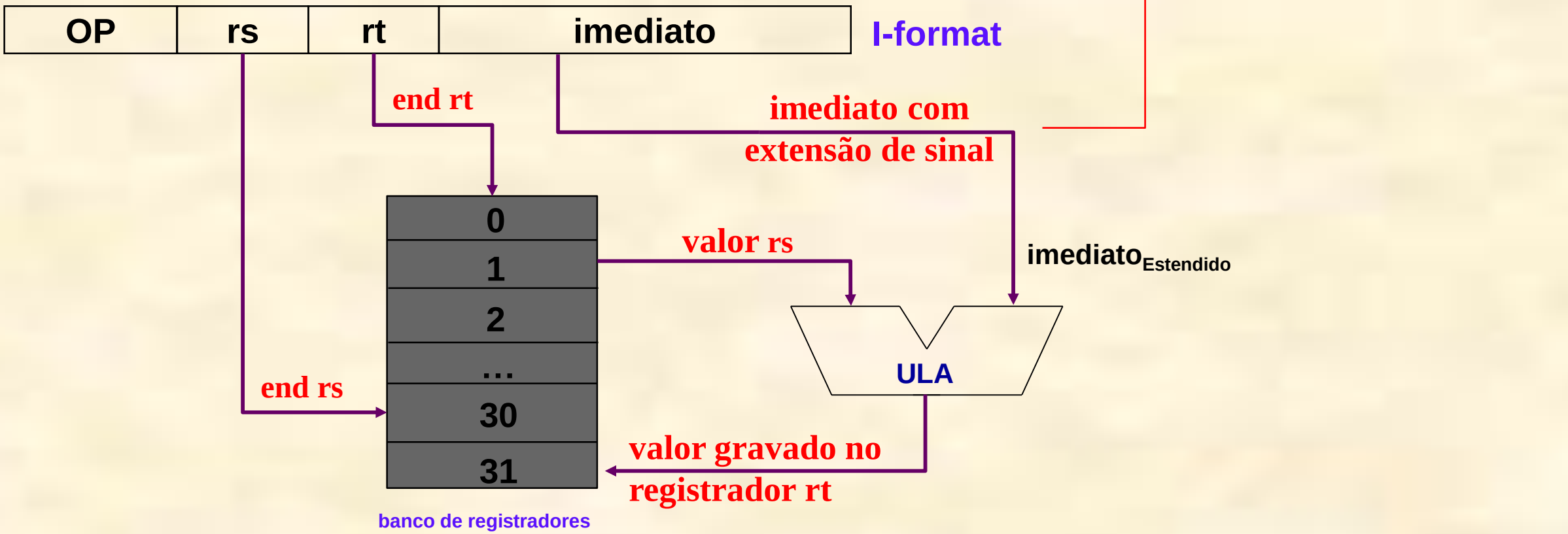
- Endereços dos registradores são indicados na própria instrução
- Exemplo: add rd, rs, rt → rd = rs + rt



Modo de Endereçamento Imediato

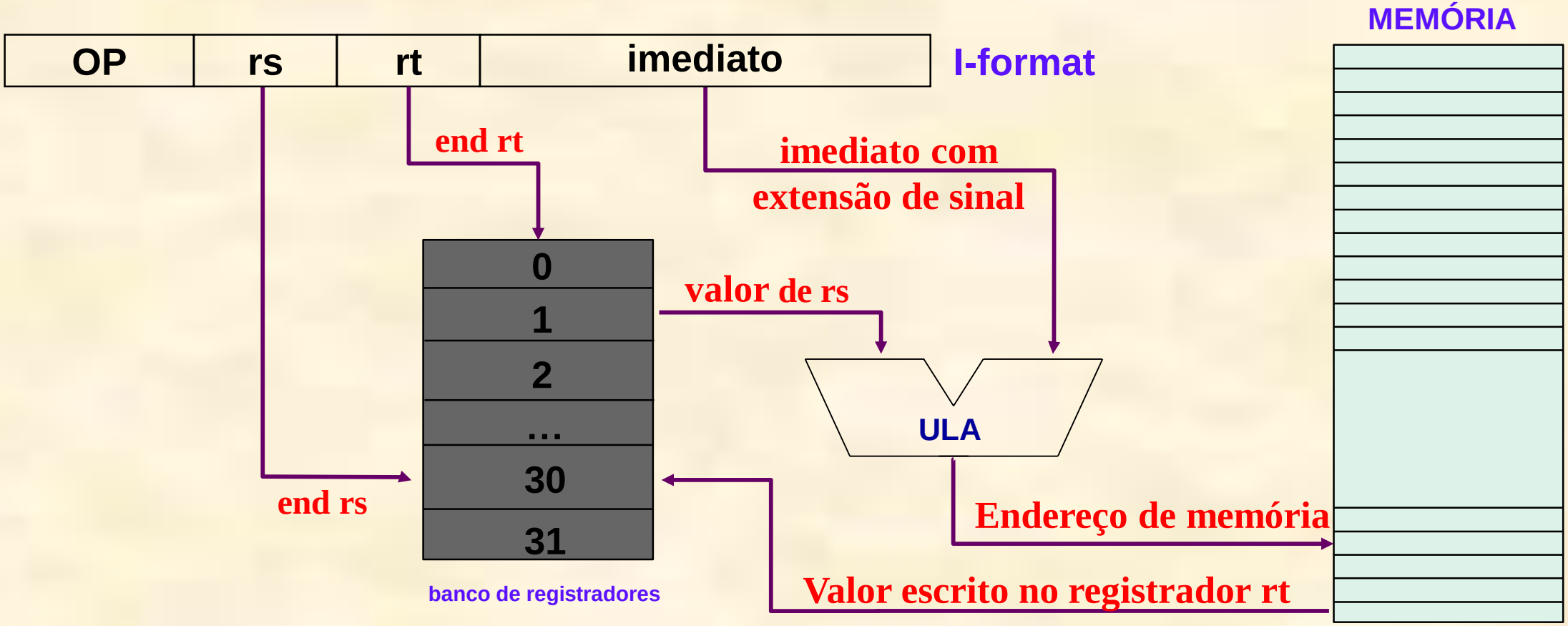
- Operações de valor de registrador com uma constante
- Exemplo: addi rt, rs, 100 → rt = rs + 100

$imediato_{Estendido}$ ← x"FFFF" & imediato when imediato(15) = '1' else x"0000" & imediato;



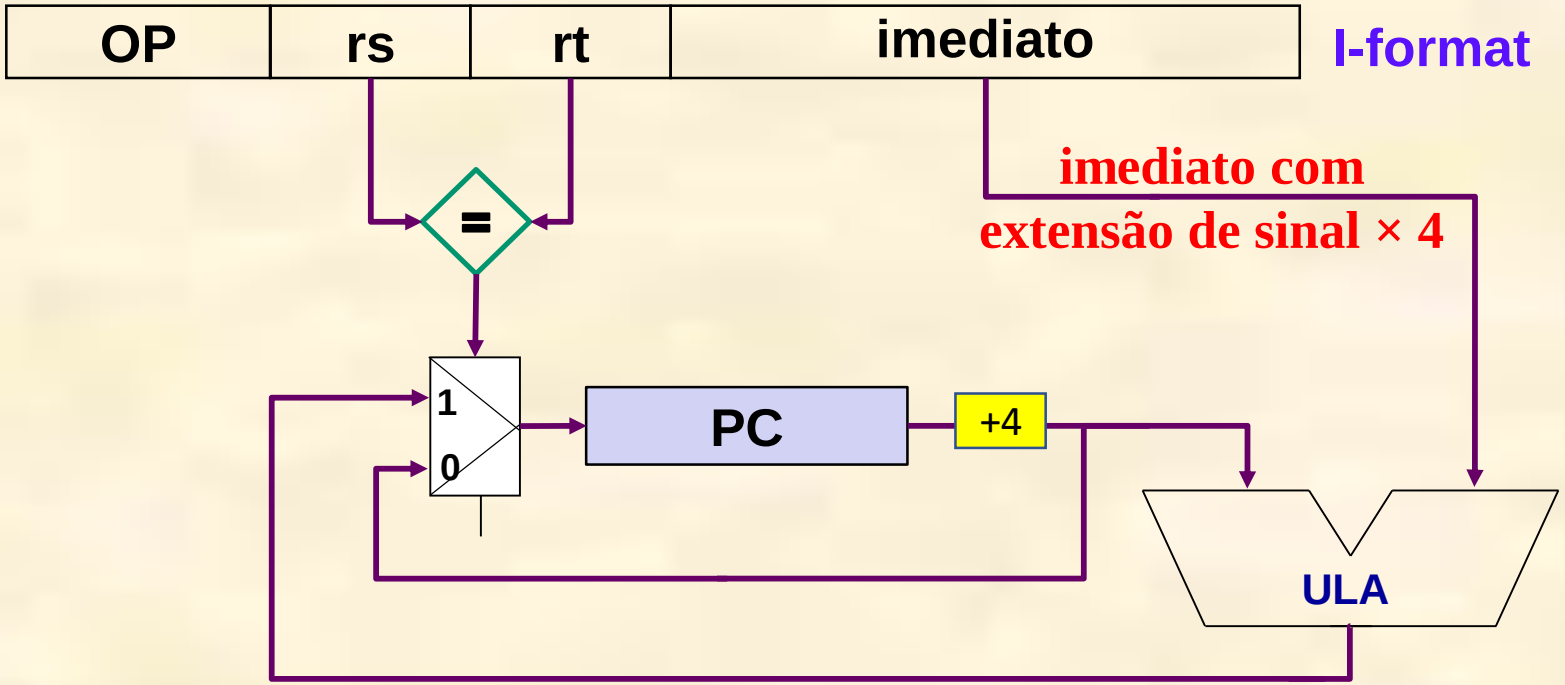
Modo de Endereçamento Base-Deslocamento

- **Acesso à memória com instruções load/store**
  - Registrador indica o endereço e a constante indica um deslocamento em relação à este endereço
- **Exemplo: lw rt, 12(rs) → rt = MEM[rs + 12]**



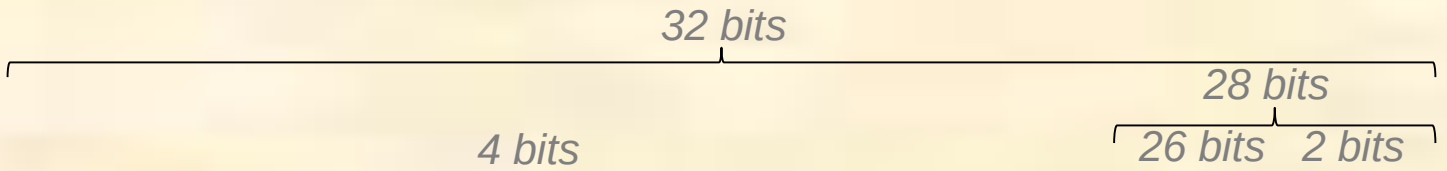
Modo de Endereçamento Relativo ao PC

- Usado em saltos condicionais
- Exemplo: beq rs, rt, valor → PC ← (rs = rt) ? PC + 4 + imediato × 4 : PC + 4

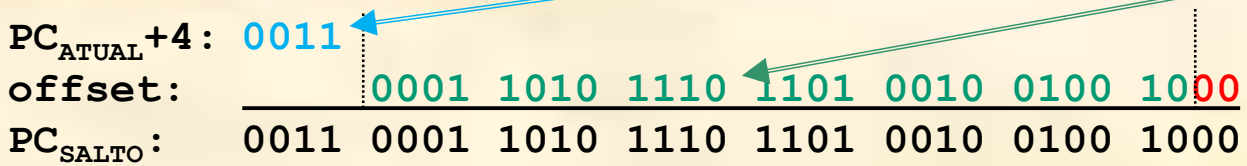


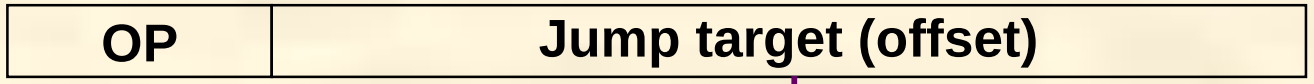
Modo de Endereçamento Direto à Imediato

- **Usado em saltos incondicionais**
- **Exemplo: j offset**



PC_{SALTO} ← 4 bits mais significativos de (PC_{ATUAL} + 4) & offset & 00

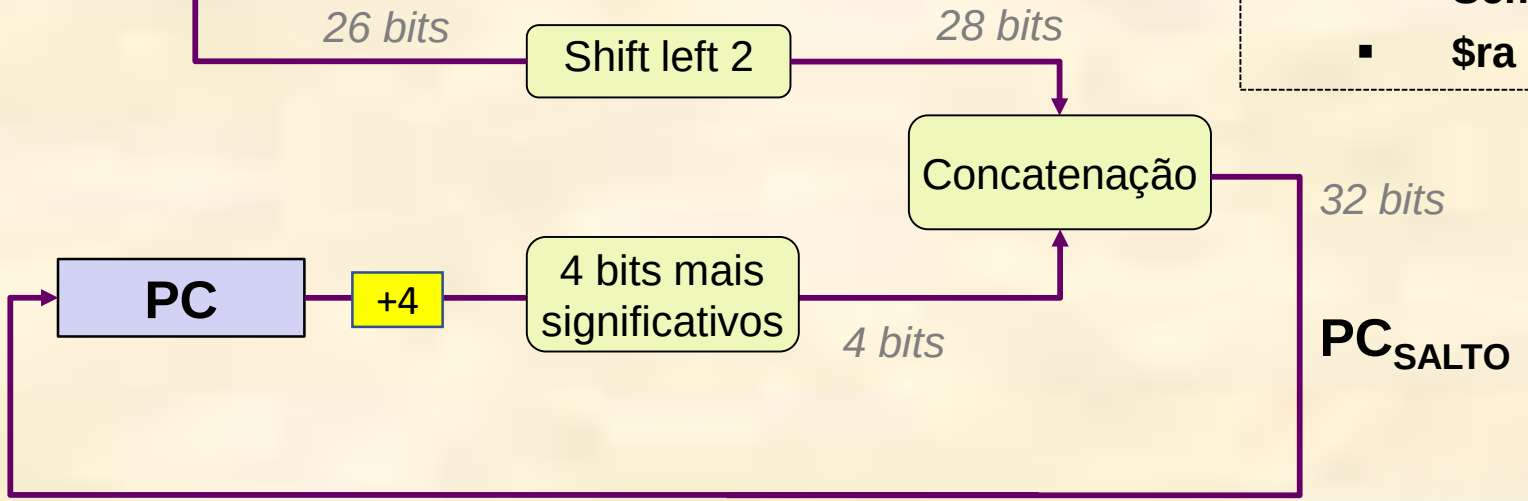




J-format

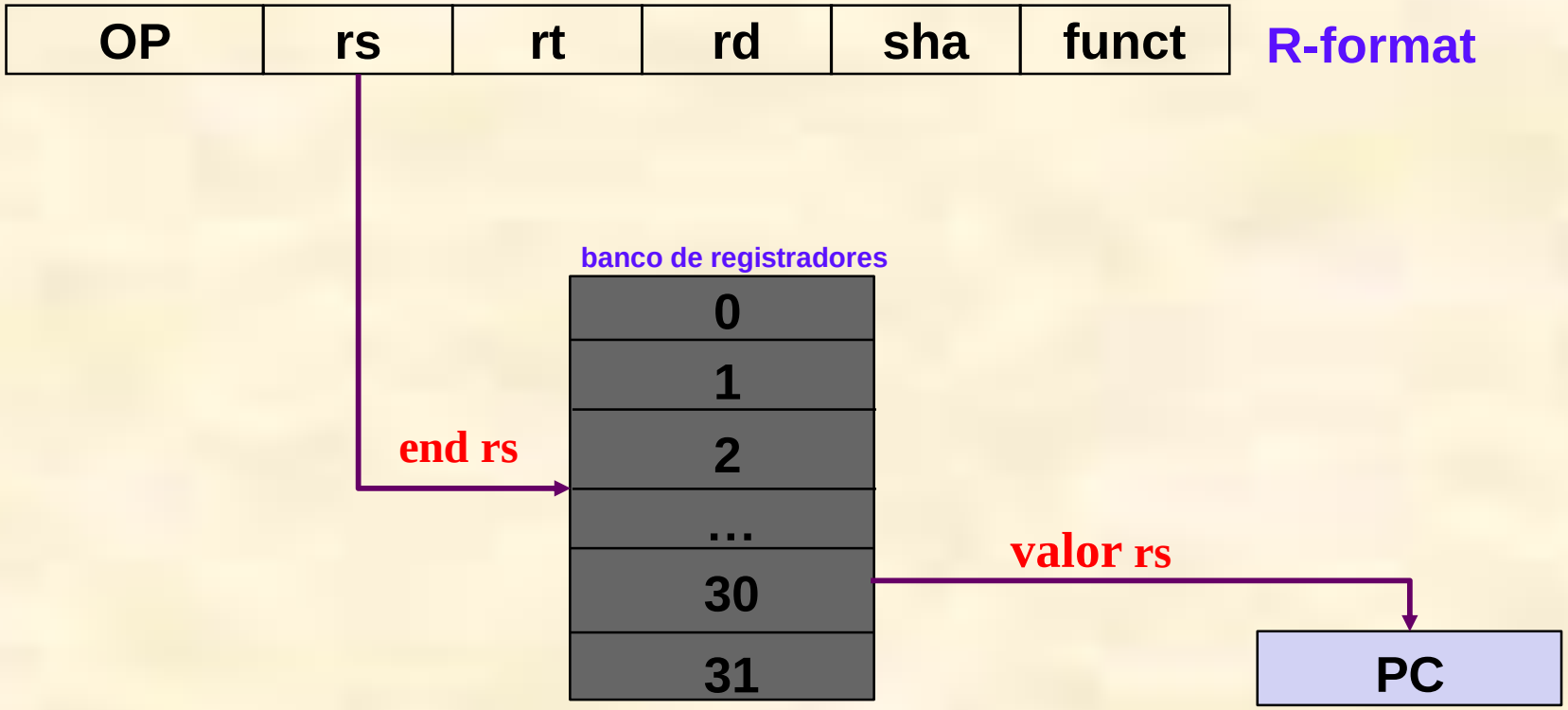
JAL – *jump* and *link*

- **Semelhante ao j**
- **$ra ← PC+4**



Modo de Endereçamento Direto à Registrador

- Usado em salto à registrador
- Exemplo: jr $ra → PC recebe o endereço de retorno



Exercícios

1. **Dados os códigos das instruções que seguem, obtenha o assembly correspondente e diga qual é o modo de endereçamento**
   a) 0x24840001
   b) 0x03e00008

2. **Descubra qual é a pseudo-instrução que implementa as instruções MIPS que seguem**
   – 0x20010001
   – 0x0024082a
   – 0x14200002

Resposta

1. **Dados os códigos das instruções que seguem, obtenha o assembly correspondente e diga qual é o modo de endereçamento**

   a) 0x24840001

- **Binário: 0010 0100 1000 0100 0000 0000 0000 0001**
  - OP: [001001] → 0x09 → addiu

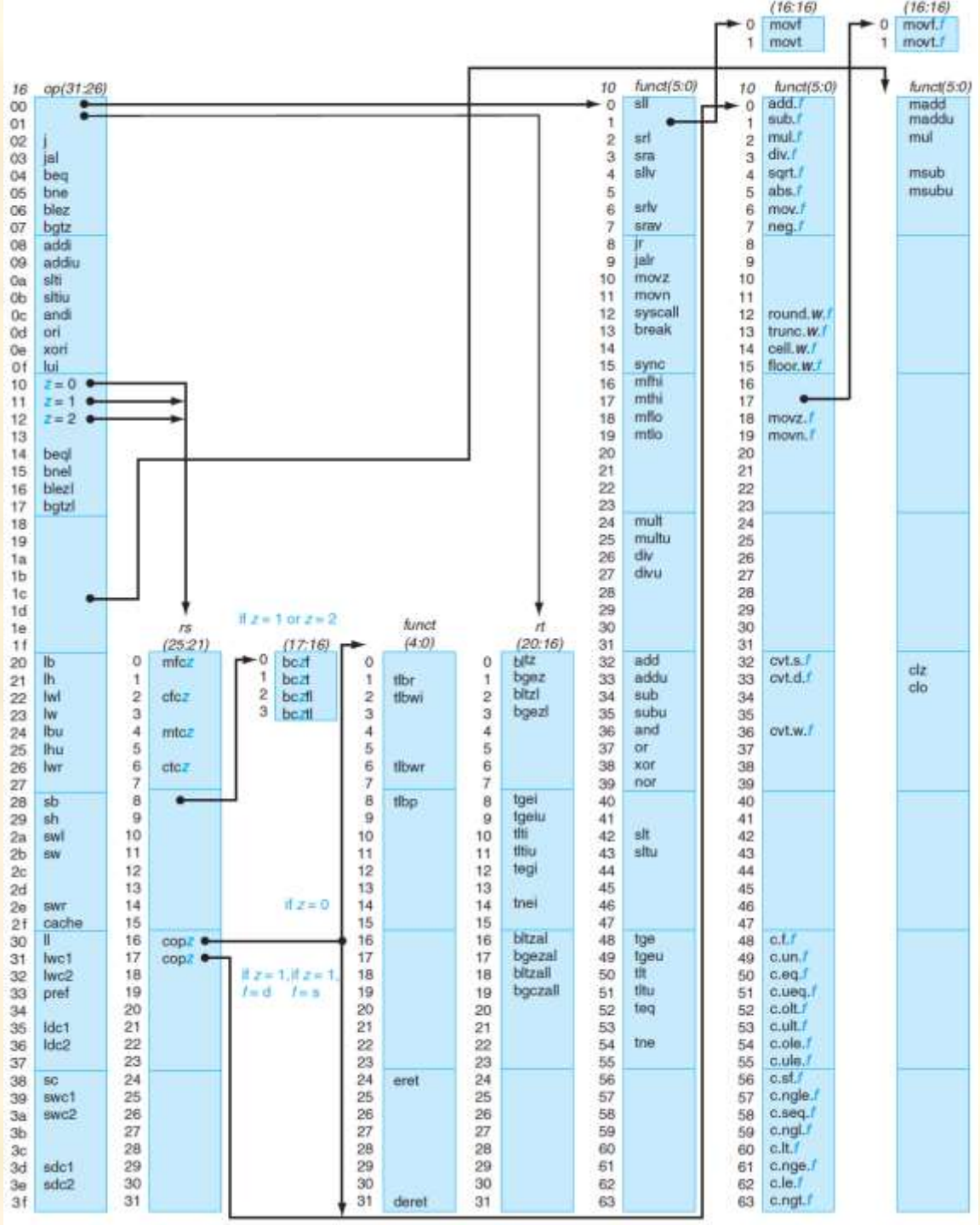
Addition immediate (without overflow)

addiu rt, rs, imm

9	rs	rt	imm
6	5	5	16

- **Agrupa: [001001][00100][00100][0000000000000001]**
- **Hexa: 0x9 0x4 0x4 0x1**
- **Identificando o assembly: addiu $a0, $a0, 1**
- **Modo de endereçamento: Imediato**

Número	Nome	Significado
0	$zero	constante 0
1	$at	reservado para o assembly
2, 3	$v0, $v1	resultado de função
4 - 7	$a0 - $a3	argumento para função
8 - 15	$t0 - $t7	temporário
16 - 23	$s0 - $s7	temporário (salvo nas chamadas de função)
24, 25	$t8, $t9	temporário
26, 27	$k0, $k1	reservado para o SO
28	$gp	apontador de área global
29	$sp	stack pointer
30	$fp	frame pointer
31	$ra	registrador de endereço de retorno



Resposta

1. **Dados os códigos das instruções que seguem, obtenha o assembly correspondente e diga qual é o modo de endereçamento**

   b) 0x03e00008

- **Binário: 0000 0011 1110 0000 0000 0000 0000 1000**
  - OP: [000000] → 0x00 → **???**
  - Funct(5:0) [001000] → 0x08 → jr

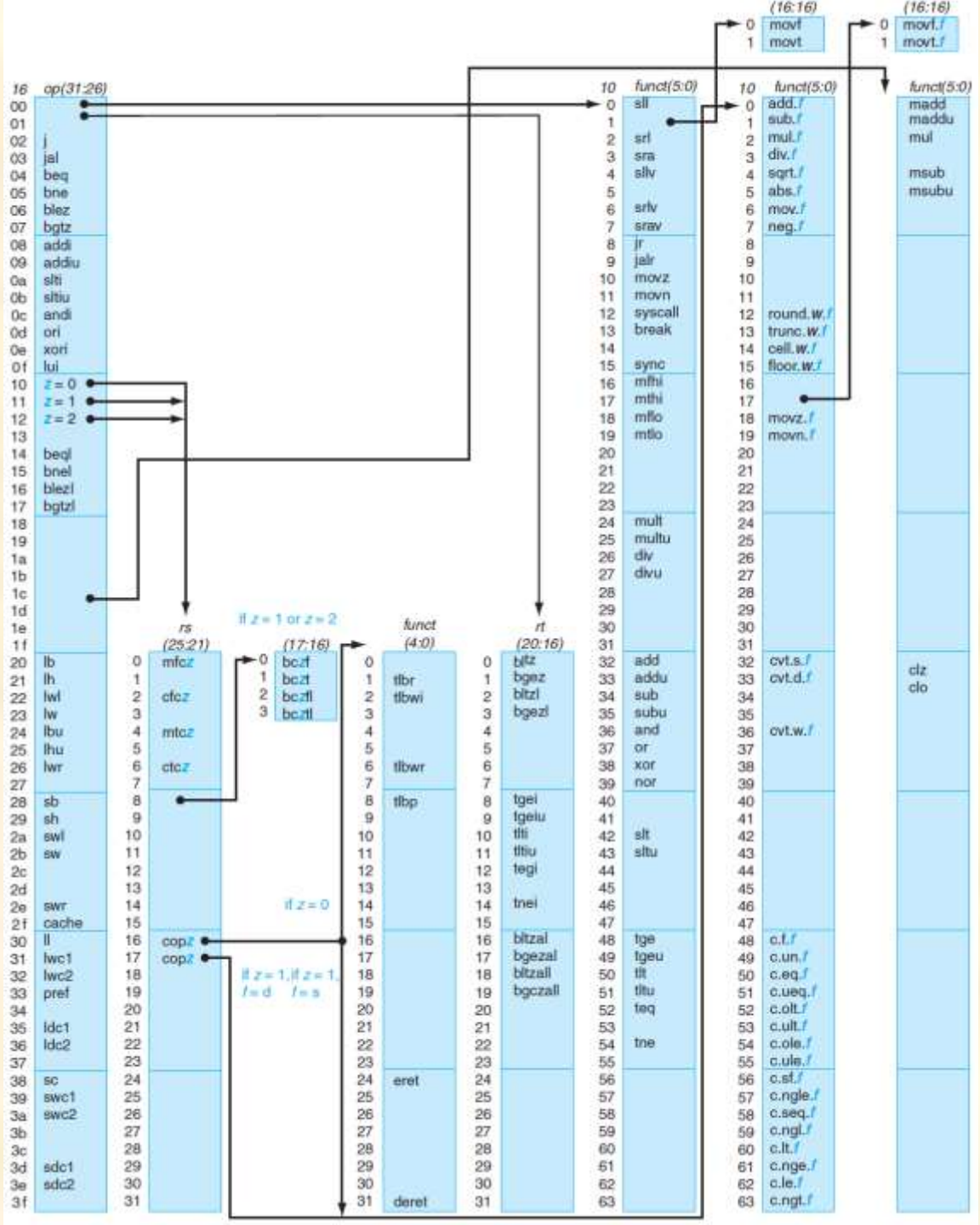
Jump register

jr rs

0	rs	0	8
6	5	15	6

- **Agrupa: [000000][11111][0000000000000000][001000]**
- **Hexa: 0x0 0x31 0x0 0x8**
- **Identificando o assembly: jr $ra**
- **Modo de endereçamento: Direto à registrador**

Número	Nome	Significado
0	$zero	constante 0
1	$at	reservado para o assembly
2, 3	$v0, $v1	resultado de função
4 - 7	$a0 - $a3	argumento para função
8 - 15	$t0 - $t7	temporário
16 - 23	$s0 - $s7	temporário (salvo nas chamadas de função)
24, 25	$t8, $t9	temporário
26, 27	$k0, $k1	reservado para o SO
28	$gp	apontador de área global
29	$sp	stack pointer
30	$fp	frame pointer
31	$ra	registrador de endereço de retorno



Resposta

2. Descubra qual é a pseudo-instrução que implementa as instruções MIPS que seguem

- 0x20010001
- 0x0024082a
- 0x14200002

- **Binário: 0010 0000 0000 0001 0000 0000 0000 0001**
  - OP: [0010000] → 0x08 → addi

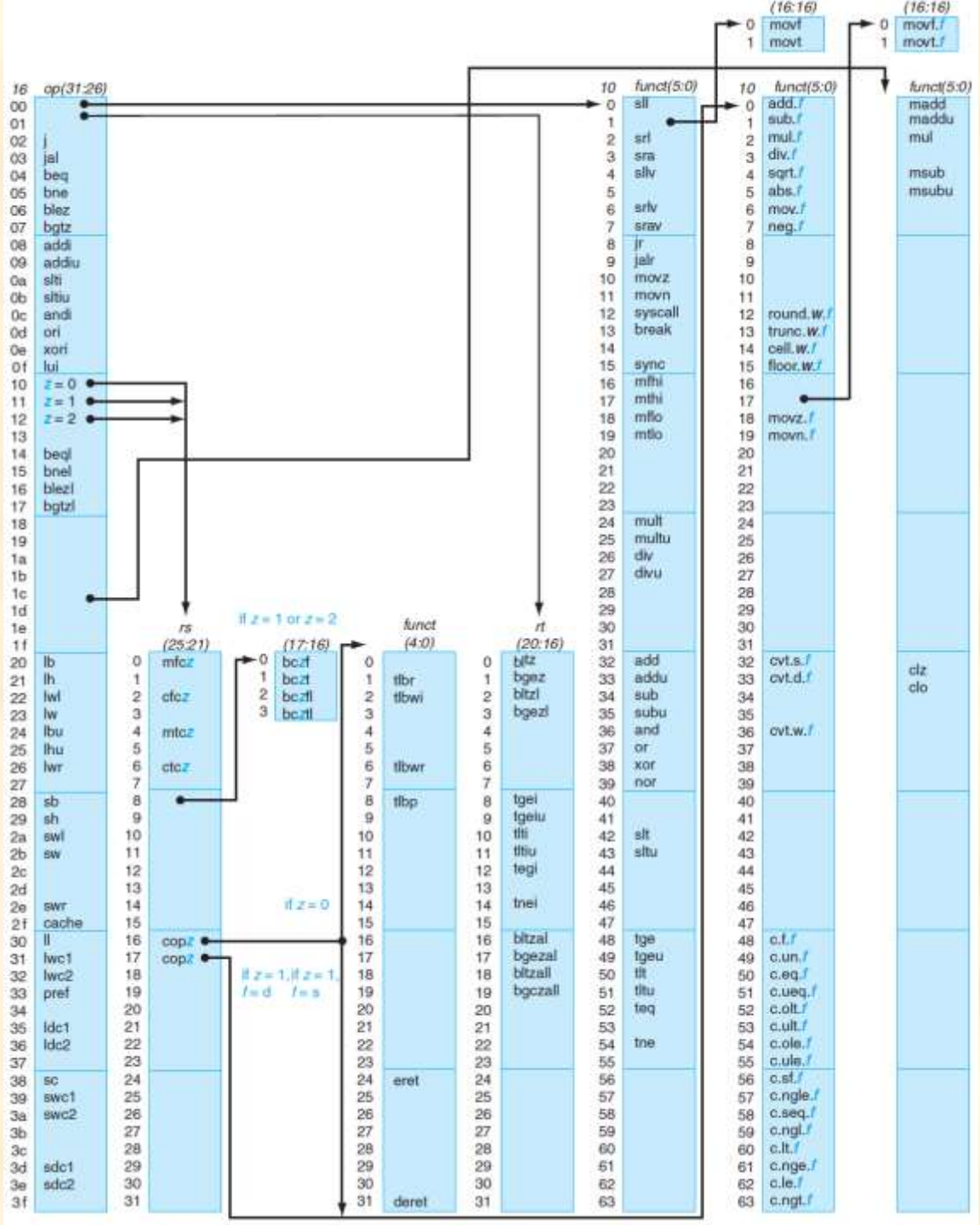
Addition immediate (with overflow)

addi rt, rs, imm

8	rs	rt	imm
6	5	5	16

- **Agrupa: [001000][00000][00001][0000000000000001]**
- **Hexa: 0x8 0x0 0x1 0x1**
- **Identificando o assembly: addi $at, $zero, 1**

Número	Nome	Significado
0	$zero	constante 0
1	$at	reservado para o assembly
2, 3	$v0, $v1	resultado de função
4 - 7	$a0 - $a3	argumento para função
8 - 15	$t0 - $t7	temporário
16 - 23	$s0 - $s7	temporário (salvo nas chamadas de função)
24, 25	$t8, $t9	temporário
26, 27	$k0, $k1	reservado para o SO
28	$gp	apontador de área global
29	$sp	stack pointer
30	$fp	frame pointer
31	$ra	registrador de endereço de retorno



Resposta

2. **Descubra qual é a pseudo-instrução que implementa as instruções MIPS que seguem**
   - 0x20010001
   - 0x0024082a
   - 0x14200002

- **Binário: 0000 0000 0010 0100 0000 1000 0010 1010**
  - OP: [000000] → 0x0 → **???**
  - Funct(5:0) [101010] → 42 → slt

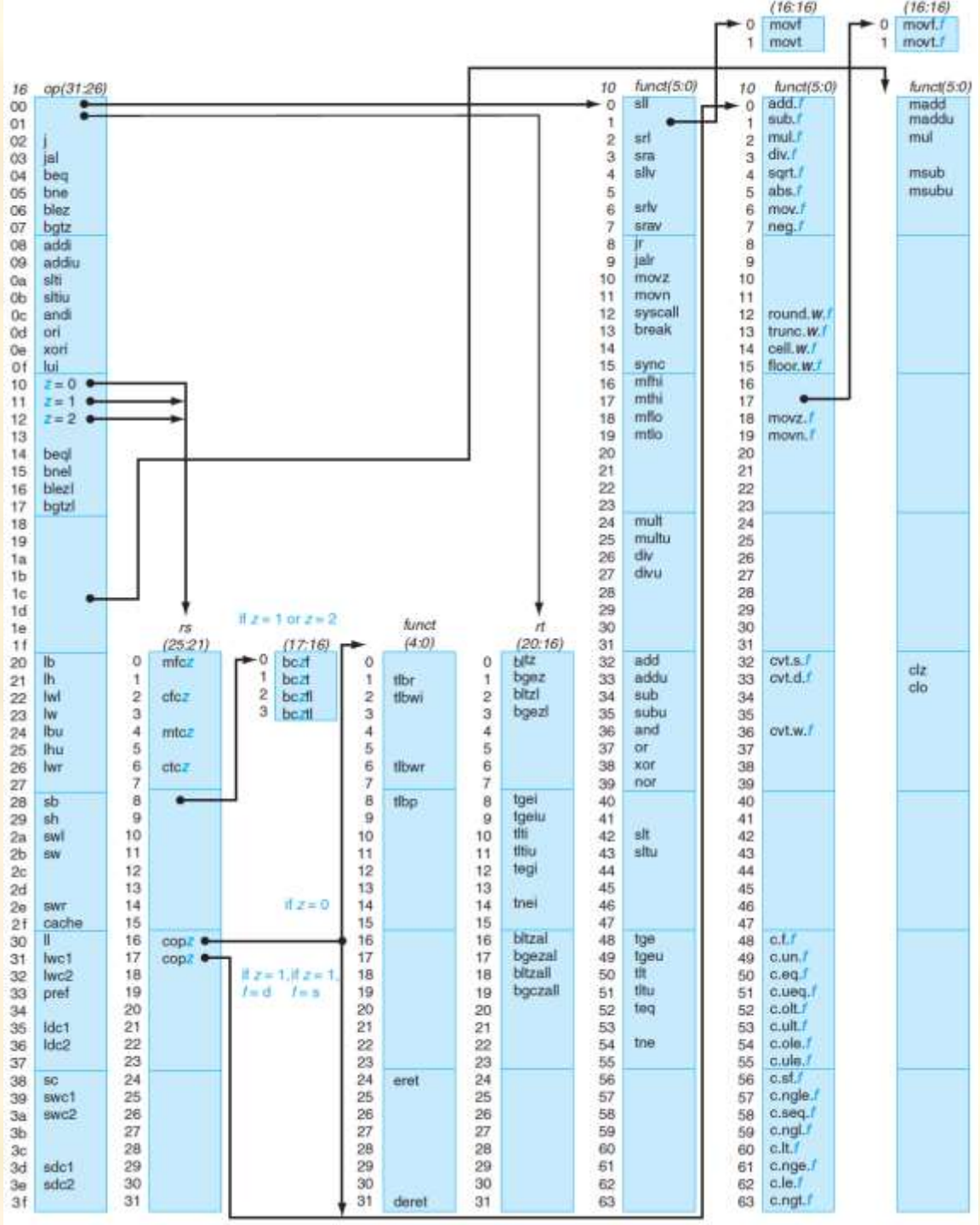
Set less than

slt rd, rs, rt

0	rs	rt	rd	0	0x2a
6	5	5	5	5	6

- **Agrup: [000000] [00001] [00100] [00001] [00000] [101010]**
- **Hexa: 0x0 0x1 0x4 0x1 0x0 0x2a**
- **Identificando o assembly: slt $at, $at, $a0**

Número	Nome	Significado
0	$zero	constante 0
1	$at	reservado para o assembly
2, 3	$v0, $v1	resultado de função
4 - 7	$a0 - $a3	argumento para função
8 - 15	$t0 - $t7	temporário
16 - 23	$s0 - $s7	temporário (salvo nas chamadas de função)
24, 25	$t8, $t9	temporário
26, 27	$k0, $k1	reservado para o SO
28	$gp	apontador de área global
29	$sp	stack pointer
30	$fp	frame pointer
31	$ra	registrador de endereço de retorno



Resposta

2. Descubra qual é a pseudo-instrução que implementa as instruções MIPS que seguem

- 0x20010001
- 0x0024082a
- 0x14200002

- **Binário: 0001 0100 0010 0000 0000 0000 0000 0010**
  - OP: [000101] → 0x05 → bne

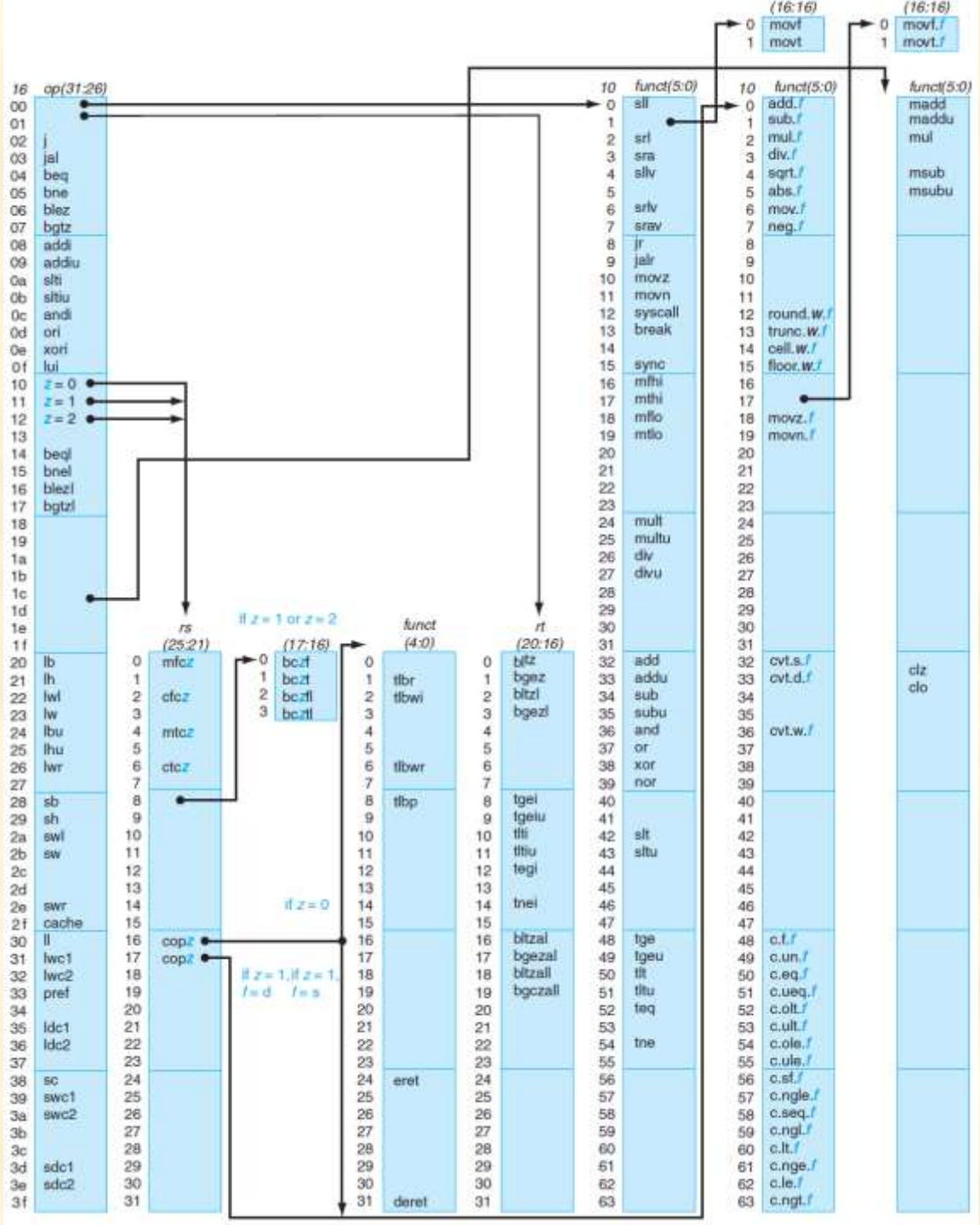
Branch on not equal

bne rs, rt, label

5	rs	rt	Offset
6	5	5	16

- **Agrupa: [000101][00001][00000][0000000000000010]**
- **Hexa: 0x5 0x1 0x0 0x2**
- **Identificando o assembly: bne $at, $zero, 2**
- **bgt $a0, 1, salto**

Número	Nome	Significado
0	$zero	constante 0
1	$at	reservado para o assembly
2, 3	$v0, $v1	resultado de função
4 - 7	$a0 - $a3	argumento para função
8 - 15	$t0 - $t7	temporário
16 - 23	$s0 - $s7	temporário (salvo nas chamadas de função)
24, 25	$t8, $t9	temporário
26, 27	$k0, $k1	reservado para o SO
28	$gp	apontador de área global
29	$sp	stack pointer
30	$fp	frame pointer
31	$ra	registrador de endereço de retorno



Resposta

2. Descubra qual é a pseudo-instrução que implementa as instruções MIPS que seguem

- 0x20010001 ➔ `addi $at, $zero, 1`
- 0x0024082a ➔ `slt $at, $at, $a0`
- 0x14200002 ➔ `bne $at, $zero, 2`

```
addi $at, $zero, 1   ➔ $at ← 1
slt  $at, $at, $a0   ➔ $at ← ($at < $a0) ? 1 : 0
                     ➔ $at ← ($a0 > 1) ? 1 : 0
bne  $at, $zero, 2   ➔ $PC ← ($at != 0) ? salta ($PC + 4 + 2 << 2) : $PC + 4
                     ➔ $PC ← ($a0 > 1) ? salta ($PC + 4 + 2 << 2) : $PC + 4
                     ➔ bgt  $a0, 1, 2
```

`bgt $a0, 1, salto`

`// salto` é um rótulo qualquer que esteja deslocado duas instruções abaixo