Universidade Federal de Pelotas
Centro de Desenvolvimento Tecnológico
Bacharelado em Ciência da Computação
Engenharia de Computação

Arquitetura e Organização de Computadores I

Prática

Aula 2

Revisão, Adição e Subtração

Prof. Guilherme Corrêa
gcorrea@inf.ufpel.edu.br

Prof. Bruno Zatt

Revisão

Assembly

- **Exemplo:**

C/C++

```
a = b & c;
```

Assembly (simbólico)

```
load b
load c
and a, b, c
store a
```

Assembly (linguagem de máquina)

```
10100010 00101011
10000100 00100010

10100011 00101011
10011000 00100110

10110010 00101011
10100100 00100011

10100001 00101011
10000000 00100000
```

Revisão

Assembly

- Vantagens
  - Desempenho
  - Sistemas embarcados
  - Tempo de execução previsível
  - Sistemas com tempo crítico
- Desvantagens
  - Programas para uma máquina específica
  - Programas longos
  - Pouca legibilidade (ou seja, comentem os seus códigos!!!)

Revisão

MIPS: estrutura básica

- Banco de registradores (32 registradores)
- Program Counter (PC)
- Memória (dados, instruções)
- Unidade Lógica e Aritmética (ULA)

Revisão

MIPS: estrutura básica

- Banco de registradores (32 registradores)
- Program Counter (PC)
- Memória (dados, instruções)
- Unidade Lógica e Aritmética (ULA)

Revisão

MIPS: Registradores

- Elemento mais alto na hierarquia de memória
- Única memória que o processador acessa diretamente
- 32 registradores de propósito geral de 32 bits
  - $0, $1, …, $31
  - operações inteiras
  - endereçamento
- $0 tem sempre valor 0
- $31 é utilizado para retorno de funções

Revisão

MIPS: Registradores

Registrador	Nome	Uso (convenção)
$0	$zero	Zero
$1	$at	*Assembler Temporary*
$2, $3	$v0, $v1	Valor de retorno de subrotina
$4 – $7	$a0 – $a3	Argumentos de subrotina
$8 – $15	$t0 – $t7	Temporários (locais à função)
$16 – $23	$s0 – $s7	Salvos (não alterados na função)
$24, $25	$t8, $t9	Temporários
$26, $27	$k0, $k1	Kernel (reservado para SO)
$28	$gp	*Global Pointer*
$29	$sp	*Stack Pointer*
$30	$fp	*Frame Pointer*
$31	$ra	Endereço de Retorno

Revisão

MIPS: Registradores

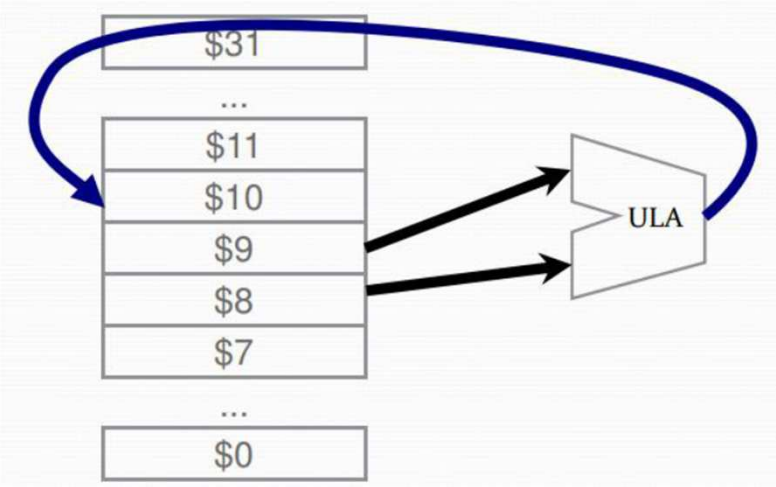
Registrador	Nome	Uso (convenção)
$0	**$zero**	**Zero**
$1	$at	*Assembler Temporary*
$2, $3	$v0, $v1	Valor de retorno de subrotina
$4 – $7	$a0 – $a3	Argumentos de subrotina
$8 – $15	**$t0 – $t7**	**Temporários (locais à função)**
$16 – $23	$s0 – $s7	Salvos (não alterados na função)
$24, $25	$t8, $t9	Temporários
$26, $27	$k0, $k1	Kernel (reservado para SO)
$28	$gp	*Global Pointer*
$29	$sp	*Stack Pointer*
$30	$fp	*Frame Pointer*
$31	$ra	Endereço de Retorno

Revisão

MIPS: Unidade Lógica e Aritmética (ULA)

- Circuito responsável pelas operações lógicas e aritméticas
- Exemplo:

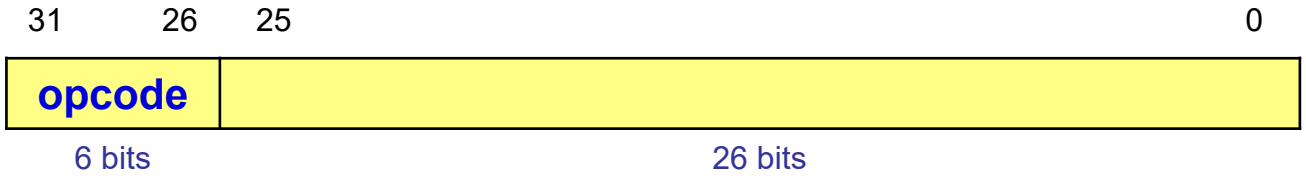
and $10, $8, $9



Revisão

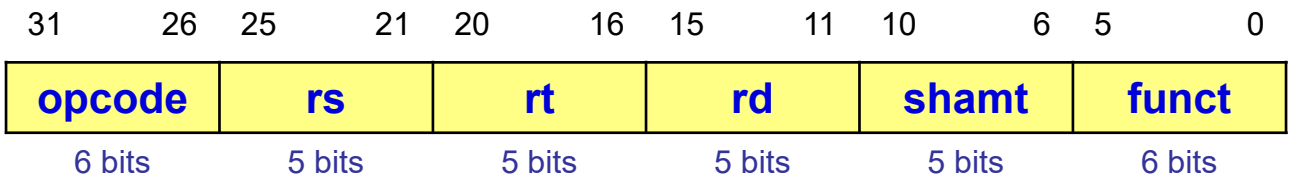
MIPS: Instruções

- Todas as instruções têm 32 bits
- Todas têm opcode de 6 bits



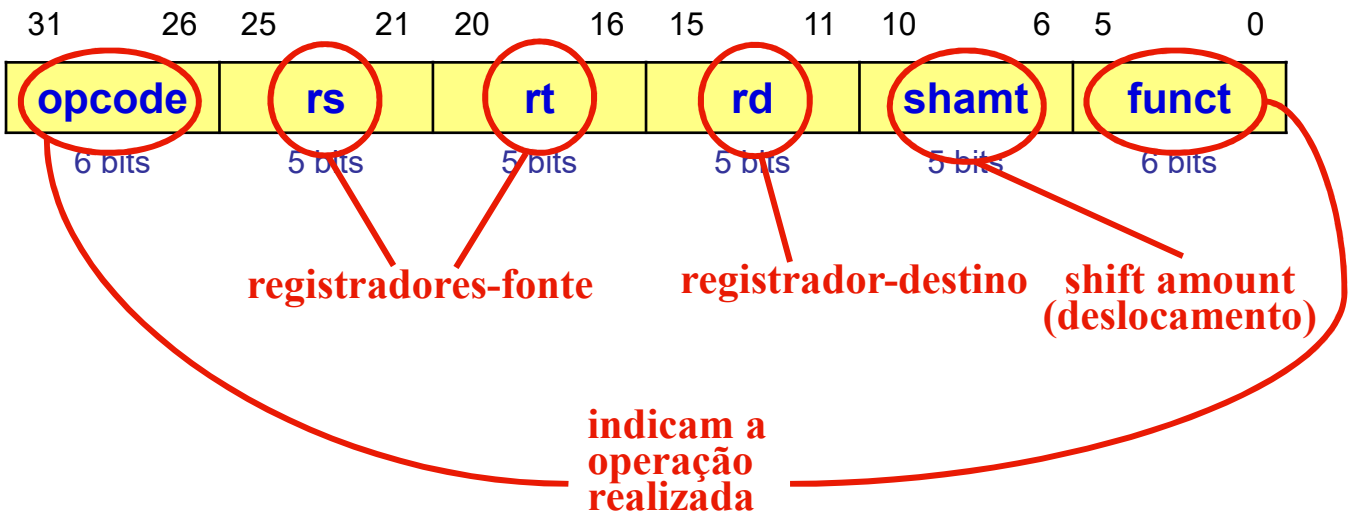
Revisão

Instruções do Tipo R (Registrador)



Revisão

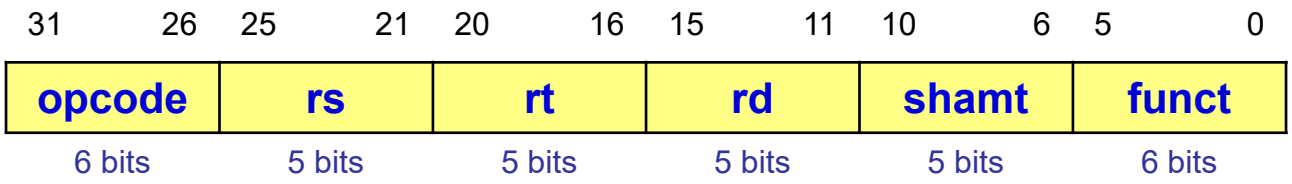
Instruções do Tipo R (Registrador)



Revisão

Operações Lógicas (and, or, xor, nor)

Tipo R



and $t1, $zero, $t3	0H	0H	BH	9H	0H	24H
or $t0, $t1, $t2	0H	9H	AH	8H	0H	25H
xor $t1, $t2, $t3	0H	AH	BH	9H	0H	26H
nor $t0, $t0, $t3	0H	8H	BH	8H	0H	27H

Revisão

Operações Lógicas (and, or, xor, nor)

Tipo R

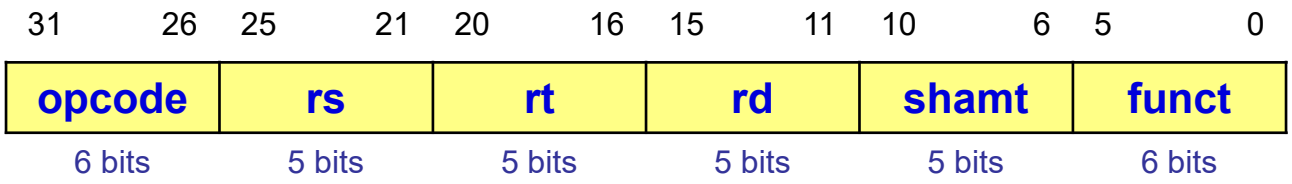
31 – 26	25 – 21	20 – 16	15 – 11	10 – 6	5 – 0
opcode	rs	rt	rd	shamt	funct
6 bits	5 bits	5 bits	5 bits	5 bits	6 bits

	opcode	rs	rt	rd	shamt	funct
and $t1, $zero, $t3	0_H	0_H	B_H	9_H	0_H	24_H
	000000	00000	01011	01001	00000	100100
or $t0, $t1, $t2	0_H	9_H	A_H	8_H	0_H	25_H
	000000	01001	01010	01000	00000	100101
xor $t1, $t2, $t3	0_H	A_H	B_H	9_H	0_H	26_H
	000000	01010	01011	01001	00000	100110
nor $t0, $t0, $t3	0_H	8_H	B_H	8_H	0_H	27_H
	000000	01011	01011	01001	00000	100111

Revisão

Operações Lógicas (sll, slr)

Tipo R



sll $t0, $t1, 12

0H	0H	9H	8H	12H	0H

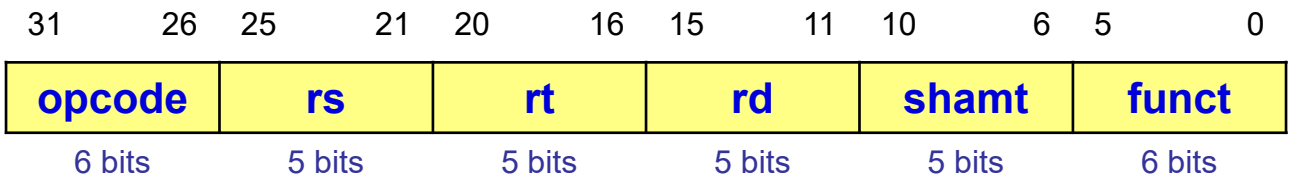
srl $t5, $t5, 28

0H	0H	DH	DH	1CH	2H

Revisão

Operações Lógicas (sll, slr)

Tipo R



sll $t0, $t1, 12

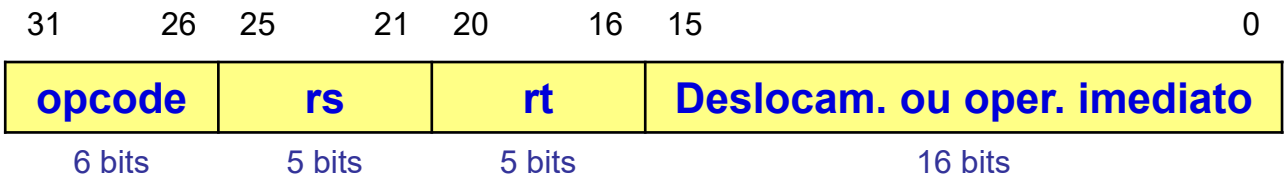
0H	0H	9H	8H	12H	0H
000000	00000	01001	01000	01100	000000

srl $t5, $t5, 28

0H	0H	DH	DH	1CH	2H
000000	00000	01101	01101	11100	000010

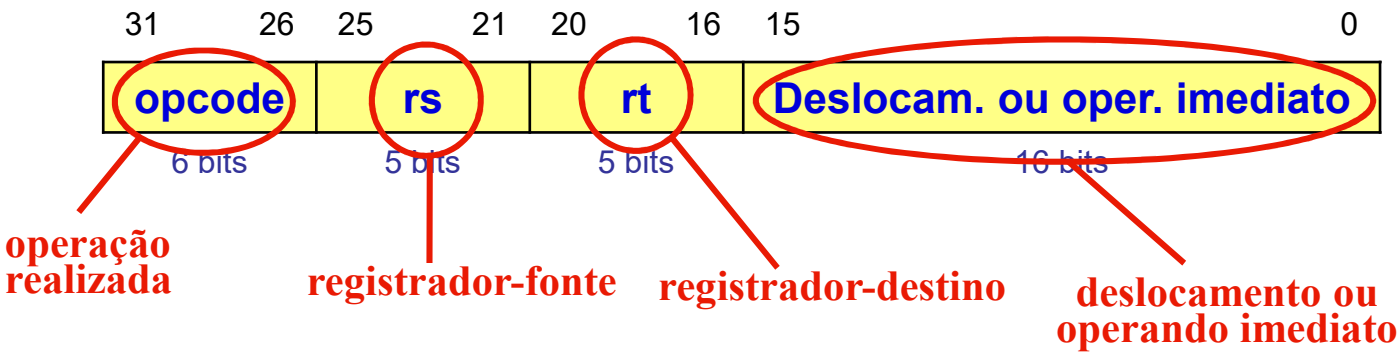
Revisão

Instruções do Tipo I (Imediato)



Revisão

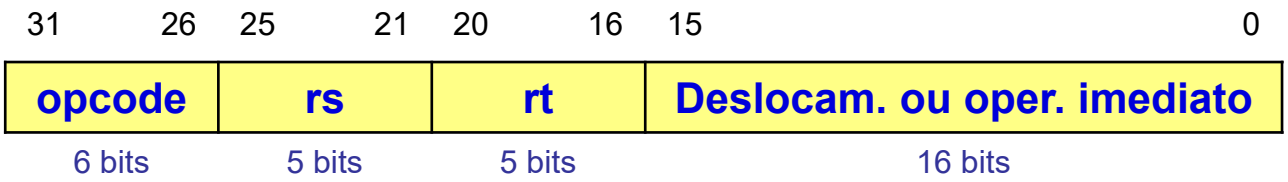
Instruções do Tipo I (Imediato)



Revisão

Operações Lógicas (andi, ori, xori)

Tipo I



andi $t1, $t0, 0xA0A0

C_H	8_H	9_H	$A0A0_H$

ori $t0, $zero, 0xFFFF

D_H	0_H	8_H	$FFFF_H$

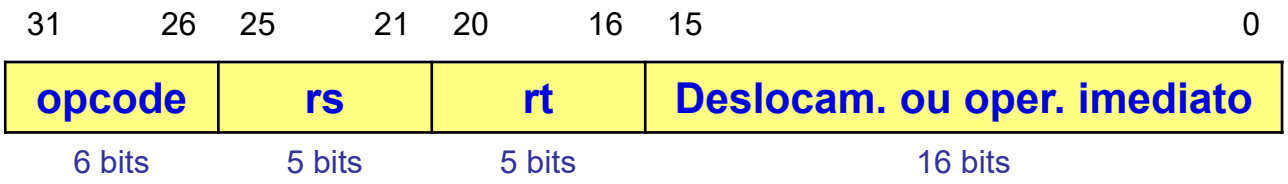
xori $t1, $t1, 0x538

E_H	9_H	9_H	538_H

Revisão

Operações Lógicas (andi, ori, xori)

Tipo I



andi $t1, $t0, 0xA0A0

C_H	8_H	9_H	$A0A0_H$
001100	01000	01001	1010000010100000

ori $t0, $zero, 0xFFFF

D_H	0_H	8_H	$FFFF_H$
001101	00000	01000	1111111111111111

xori $t1, $t1, 0x538

E_H	9_H	9_H	538_H
001110	01001	01001	0000010100111000

Revisão

Utilizações Especiais (or, ori, nor, sll)

or $t1, $t2, $zero	Copia em $t1 o valor de $t2
ori $t1, $zero, 0x6	Copia em $t1 o valor 0x6
nor $t1, $t2, $zero	Copia em $t1 o valor de $t2 negado
sll $t1, $t0, 4	Coloca em $t1 o valor de $t0 multiplicado por $2^4 = 16$

Exemplo:

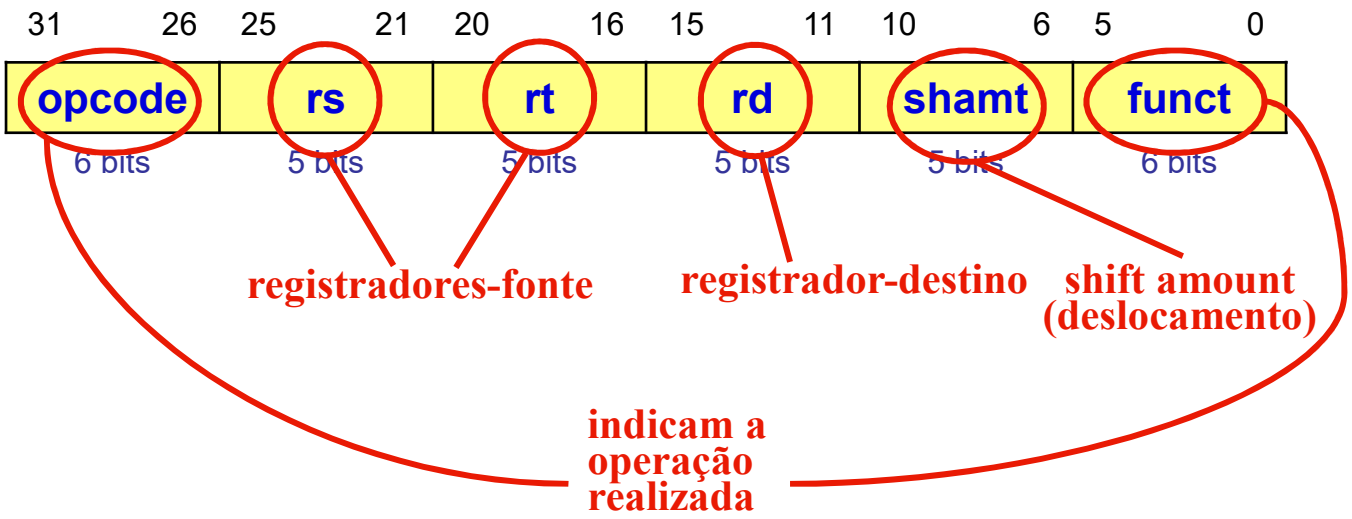
$t0 contém $A_H = 00000000000000000000000000001010_B = 10_D$

Após sll $t1, $t0, 4

$t1 contém $A0_H = 00000000000000000000000010100000_B = 160_D$

Operações Aritméticas

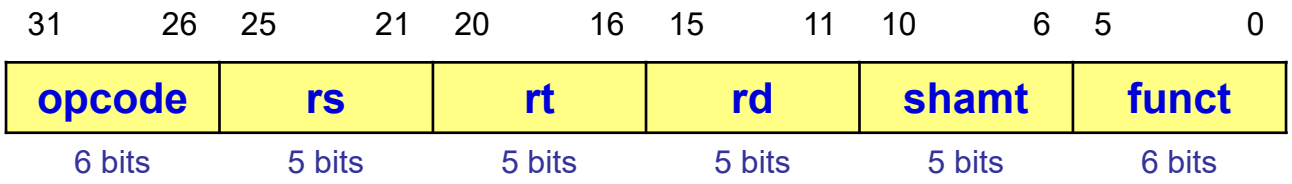
Instruções do Tipo R (Registrador)



Operações Aritméticas

Soma, Subtração (add, sub)

Tipo R



add $t2, $t0, $t1

0H	8H	9H	AH	0H	20H

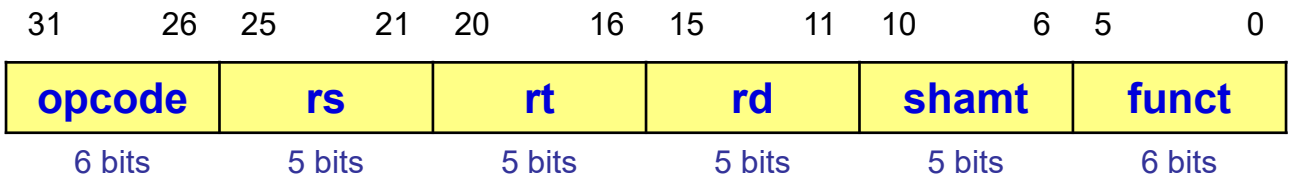
sub $t2, $t0, $t1

0H	8H	9H	AH	0H	22H

Operações Aritméticas

Soma, Subtração (add, sub)

Tipo R



add $t2, $t0, $t1

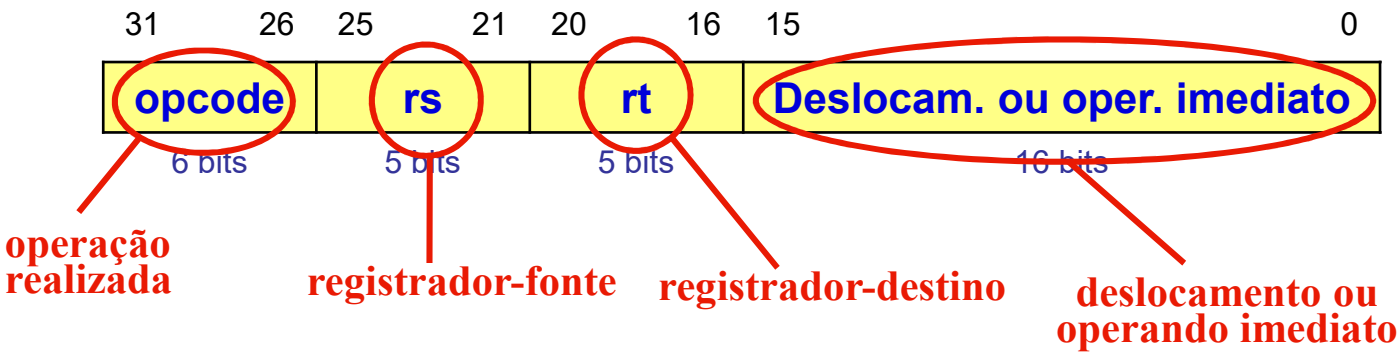
0H	8H	9H	AH	0H	20H
000000	01000	01001	01010	00000	100000

sub $t2, $t0, $t1

0H	8H	9H	AH	0H	22H
000000	01000	01001	01010	00000	100010

Operações Aritméticas

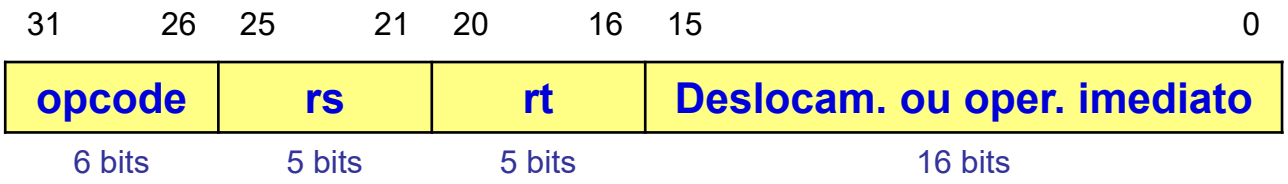
Instruções do Tipo I (Imediato)



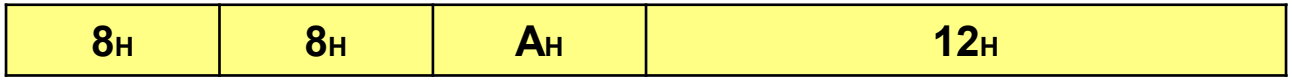
Operações Aritméticas

Soma Imediata (addi)

Tipo I



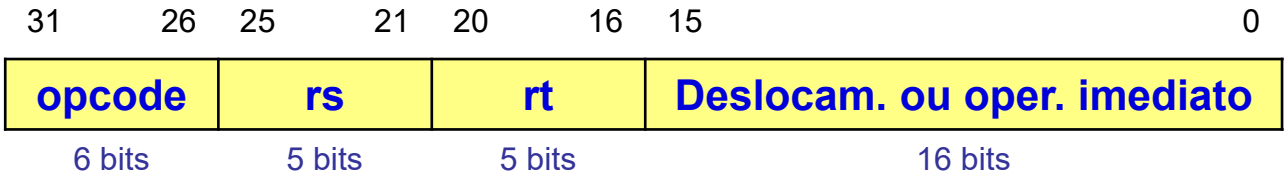
addi $t2, $t0, 0x12



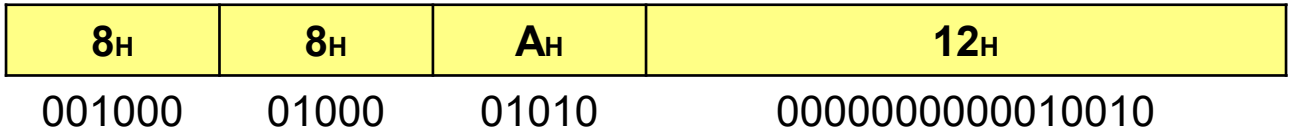
Operações Aritméticas

Soma Imediata (addi)

Tipo I



addi $t2, $t0, 0x12



Por que não existe instrução para subtração imediata? (subi)

Operações Aritméticas

Por que não existe instrução para subtração imediata? (subi)

- Constantes negativas aparecem com muito menos frequência
- O campo imediato de instruções do Tipo I também mantém constantes negativas
- Ou seja: a soma imediata utilizando uma constante negativa é o mesmo que uma subtração imediata com uma constante positiva
- Portanto: subtrações imediatas devem ser feitas com addi !

$$a - b == a + (-b)$$

Operações Aritméticas

Inteiros Positivos: Representação

- Quantos inteiros positivos são representáveis com N bits?
- Resposta: de 0 a 2^N-1
- Exemplo: $N = 8$

```
00000000 = 0
00000001 = 2^0 = 1
00000010 = 2^1 = 2
00000011 = 2^1+2^0 = 2+1 = 3
00000100 = 2^2 = 4
00000101 = 2^2+2^0 = 4+1 = 5
...
11111111 = 2^7+2^6+2^5+2^4+2^3+2^2+2^1+2^0 = 255 = 2^N-1
```

- Se $N = 16$, representação de 0 a 65.535
- Se $N = 32$, representação de 0 a 4.294.967.295

Operações Aritméticas

Inteiros Positivos: *Overflow*

- **Em somas de inteiros positivos, o overflow acontece quando o bit de *carry* mais significativo ("vai-um") é 1.**

```
  0 0 1 1 1 1 0 0          1 0 0 0 0 0 0 0
    00101100   44            10000000   128
+   00110111 + 55        +   10000000 + 128
------------   --        ------------   ---
    01100011   99            00000000   ~~256~~ 0
```

Sem *overflow* **Com *overflow***

Operações Aritméticas

Complemento de Dois

- Serve para representar inteiros positivos e negativos
- Bit mais significativo:
  - Inteiros positivos: 0
  - Inteiros negativos: 1
- Como se calcula?

  $-X == \text{NOT}(X) + 1$

  Isto é, o número negativo é representado pelo mesmo número positivo invertido, somado em uma unidade
- Exemplo (para um número representado com 4 bits):

  $5_D = 0101_B$

  $-5_D = \text{NOT}(0101_B)+0001_B = 1010_B+0001_B = 1011_B$

Operações Aritméticas

Complemento de Dois

- **Conveniente porque a soma simples entre números positivos e negativos em Complemento de Dois funciona corretamente**
- **Exemplo:**

```
  00001000          44
+ 11001001      + (-55)
----------      --------
  11110101        (-11)
```

Operações Aritméticas

Complemento de Dois: Representação

- Quantos **inteiros** são representáveis com N bits?
- Resposta: de $-2^{(N-1)}$ a $2^{(N-1)}-1$
- Exemplo: $N = 8$

```
10000000 → 01111111 → 01111111 + 1 = 10000000 = -128
10000001 → 01111110 → 01111110 + 1 = 01111111 = -127
…
11111111 → 00000000 → 00000000 + 1 = 00000001 = -1
00000000 = 0
00000001 = 1
…
00000101 = 5
…
01111111 = 127 = 2^(N-1)-1
```

Descobrimos o valor de um número em complemento de dois calculando novamente o complemento de dois.

- Se $N = 32$, representação de -2.147.483.648 a 2.147.483.647

Operações Aritméticas

Complemento de Dois: *Overflow*

- **Em somas com números em complemento de dois, o overflow acontece quando os dois bits de *carry* mais significativos ("vai-um") são diferentes.**

```
  11111111          63          01111100          63
+ 00111111      +   (-43)     +  00111111      +  100
  --------          ----        --------          ---
  00010100          20          10100011          ~~163~~ (-93)
```

Sem *overflow* **Com *overflow***

Operações Aritméticas

Instruções Especiais: addu, addiu e subu

- **Realizam a soma (ou subtração) sobre inteiros sem sinal (*unsigned*)**
  - **Ou seja, sabemos o resultado final sem necessidade de realizar o Complemento de Dois**
- **Não causam exceção (*trap*) em caso de *overflow***
- **O usuário deve evitar operações que causem *overflow***