GSI013–Arq. e Org. de Computadores

Operações Aritméticas e Lógicas

Universidade Federal de Uberlândia
Faculdade de Computação
Prof. Dr. rer. nat. Daniel D. Abdala

Na Aula Anterior ...

- A linguagem Assembly;
- Montadores;
- Ligadores;
  - Ligação Estática;
  - Ligação Dinâmica;
- Carregadores;
- Algumas palavras sobre Compiladores;
- Otimização de código;
- Programas executáveis;

Nesta Aula

- Instruções aritméticas em $\mathbb{Z}$;
- Formato e Codificação de Instruções;
- Overflow e underflow;
- Instruções aritméticas em $\mathbb{R}$;
- Instruções lógicas;

Instruções Aritméticas ($\mathbb{N}$ e $\mathbb{Z}$)

OP	Descrição	Exemplo
add	Adição com trap no overflow (não suportado pelo C)	add $rd, $rs, $rt
addi	Adição com trap no overflow – segundo operando cte.	addi $rt, $rs, cte
addiu	Adição sem trap no overflow – segundo operando cte.	addi $rt, $rs, cte
addu	Adição sem trap no overflow	addu $rd, $rs, $rt
aui	Adiciona a cte. Aos bits mais significativos	aui $rt, $rs, cte
clo	Conta # de zeros até achar um 1	clo $rd, $rs
clz	Conta # de uns até achar um 0	clz $rd, $rs
div	Divisão com overflow (LO ← quociente | HI ← resto)	div $rs, $rt
divu	Divisão Natural sem overflow	divu $rs, $rt
madd	Multiplica e adiciona	madd $rs, $rt
maddu	Multiplica e adiciona (Naturais)	maddu $rs, $rt

Instruções Aritméticas ($\mathbb{N}$ e $\mathbb{Z}$)

OP	Descrição	Exemplo
mfhi	Move o dado contido no registrador HI	mfhi $rs
mflo	Move o dado contido no registrador LO	mflo $rs
msub	Multiplica e subtrai	msub $rs, $rt
msubu	Multiplica e subtrai (Naturais)	msubu $rs, $rt
mtlo	Move o dado contido em $rs para o registrador LO	mtlo $rs
mthi	Move o dado contido em r$ para o registrador HI	mthi $rs
mul	Multiplicação s/ overflow	mul $rd, $rs, $rt
mult	Multiplicação (hi $\leftarrow$ 32bits$_{MSB}$ | lo $\leftarrow$ 32bits$_{LSB}$)	mult $rs, $rt
multu	Multiplicação Natural (hi $\leftarrow$ 32bits$_{MSB}$ | lo $\leftarrow$ 32bits$_{LSB}$)	multu $rs, $rt
sub	Subtração com overflow	sub $rd, $rs, $rt
subu	Subtração sem overflow	sub u $rd, $rs, $rt

Exemplo

```
⋮
long a,b,c,d = 42;
d = (a+b) – (a+c);
⋮
```

a → $s0
b → $s1
c → $s2
d → $s3

```
add $t0, $s0, $s1
add $t1, $s0, $s2
sub $s3, $t0, $t1
```

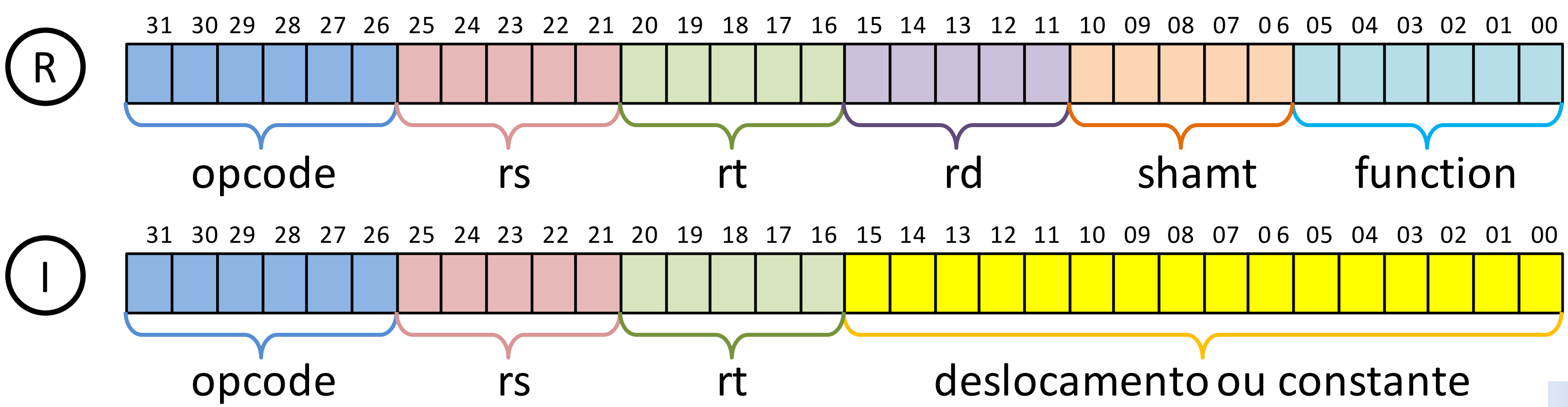
```
##############################################################
#instrucoesAritmeticas.m
#
#DDA 21.08.2016
##############################################################
.data
va:    .word 42
vb:    .word 10
vc:    .word 0x000000FF
vd:    .word 0xFFF00000
#-------------------------------------------------------------
.text

#carrega variáveis
la     $s0, va
lw     $s0, 0($s0)
la     $s1, vb
lw     $s1, 0($s1)
la     $s2, vc
lw     $s2, 0($s2)
la     $s3, vd
lw     $s3, 0($s3)

#add e suas variantes
```

```
add   $t0, $s0, $s1
addi  $t1, $s0, 42
addiu $t2, $s0, 42
addu  $t3, $s0, $s1
#aui   $t4, 0xFFFF <- não implementado no MARS

#contagem de zeros e uns iniciais
clo   $t4, $s3
clz   $t5, $s2

#divisão
div   $s0, $s1
mflo  $t6       #<- geralmente duas instruções depois do div
mfhi  $t7
divu  $s0, $s1
mflo  $t0       #<- geralmente duas instruções depois do div
mfhi  $t1

madd  $s0, $s1 #hilo += (long long) s * (long long) t
maddu $s0, $s1 #hilo += (long long) s * (long long) t (wo overflow)

mtlo  $s1
mthi  $s1
msub  $s0, $s1 #hilo -= (long long) s * (long long) t
msubu $s0, $s1 #hilo -= (long long) s * (long long) t (wo overflow)
```

```

mul   $t0,$s0, $s1
mult  $s0, $s1
multu $s0, $s1

sub   $t0, $s0, $s1
subu  $t3, $s0, $s1

#finaliza o programa
li    $v0, 10
syscall
#------------------------------------------------------------
```

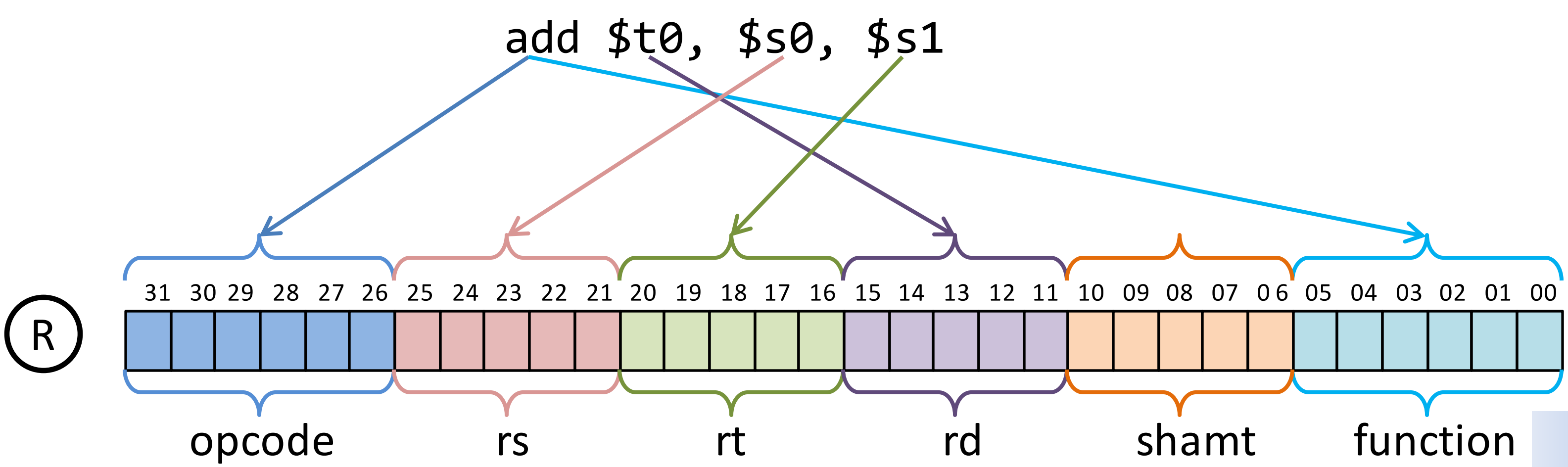
Formato das Instruções Aritméticas e Lógicas

- No MIPS todas as Instruções possuem 32 bits, sempre!
- Faz-se necessário utilizar um código binário para codificar as instruções em 32 bits;
- A maioria das instruções aritméticas e lógicas são codificadas usando o formato R, as remanescentes utilizando o formato I;
- Para interpretar o padrão de bits, o hardware utiliza as seguintes máscaras de bits:



Codificação

- Cada campo de uma instrução é codificado como uma sequência de bits;
- A maioria das instruções é do tipo R, pois todos os operadores são registradores;
- Para codificar qual dos 32 registradores são necessário 5 bits ($2^5 = 32$) (campos rs, rt e rd);
- Opcode codifica qual instrução. O campo function é usado para aumentar o número de instruções codificáveis;
- Por fim o campo shamt (shift ammount) é usado apenas nas instruções de deslocamento, e zerado nas demais;



Exemplo

```
⋮
long a,b,c,d,e,f = 42;
a = ((b*c) / (d*e)) – ((e+f)*(e-f))
⋮
```

a → $s0
b → $s1
c → $s2
d → $s3
e → $s4
f → $s5

```
mul  $t0, $s1, $s2
mul  $t1, $s3, $s4
add  $t2, $s4, $s5
sub  $t3, $s4, $s5
div  $t0, $t1
mflo $t4
mul  $t5, $t2, $t3
sub  $s0, $t4, $t5
```

```
mul  $t0, $s1, $s2
mul  $t1, $s3, $s4
div  $t0, $t1
mflo $t0
add  $t1, $s4, $s5
sub  $t2, $s4, $s5
mul  $t1, $t1, $t2
sub  $s0, $t0, $t1
```

Outros usos para o ADD/ADDI

- Inicializar o valor de um registrador;
  - addi $s0,$zero,42
- Zerar um registrador
  - add $s0,$zero,$zero
  - addi $s0,$zero,$zero
- Mover dados de um registrador para outro;
  - add $s0,$zero,$s1
- Incremento;
  - addi $s0,$s0,1
- Decremento;
  - addi $s0,$s0,-1

A família do ADD

OP	Descrição	Exemplo
add	Adição com overflow	add $rd,$rs,$rt
addi	Adição com overflow – segundo operando constante	addi $rt,$rs,cte
addu	Adição sem overflow de números não sinalizados	addu $rd,$rs,$rt
addiu	Adição sem overflow de números não sinalizados – segundo operando constante	addiu $rt,$rs,cte

- Qual a diferença entre “com overflow” e “sem overflow”?
  - Problema com a nomenclatura
  - Simplesmente significa que o sinal de overflow, quando ocorrer não lançará uma exceção.

Complemento de 2

- O complemento de 2 é calculado pela inversão de cada um dos bits do número. Subsequentemente soma-se 1 ao valor dos bits invertidos;

$$
\begin{array}{rl}
 & 0\,0\,1\,0 \Rightarrow +2_{10} \\
 & 1\,1\,0\,1 \\
+ & 0\,0\,0\,1 \\
\hline
 & 1\,1\,1\,0 \Rightarrow -2_{10}
\end{array}
$$

Decimal	Comp. 2
7	0111
6	0110
5	0101
4	0100
3	0011
2	0010
1	0001
0	0000
-1	1111
-2	1110
-3	1101
-4	1100
-5	1011
-6	1010
-7	1001
-8	1000

Overflow

- Qual a diferença entre uma instrução com e sem overflow?

$$\begin{array}{rcccl} & 0 & 1\,1\,1\,1\,1\,1\,1 & \Rightarrow & 127_{10} \\ + & 0 & 0\,0\,0\,0\,0\,0\,1 & \Rightarrow & 001_{10} \\ \hline & 1 & 0\,0\,0\,0\,0\,0\,0 & \Rightarrow & -001_{10} \end{array}$$

Underflow

- Qual a diferença entre uma instrução com e sem overflow?

	1	0 0 0 0 0 0 0	⇨	-128_{10}
-	0	0 0 0 0 0 0 1	⇨	-001_{10}

	1	0 0 0 0 0 0 0	⇨	-128_{10}
+	1	1 1 1 1 1 1 1	⇨	-001_{10}
	0	1 1 1 1 1 1 1	⇨	$+127_{10}$

	0	0 0 0 0 0 0 1
	⇩	
	1	1 1 1 1 1 1 0
+	0	0 0 0 0 0 0 1
	1	1 1 1 1 1 1 1

Multiplicação de Números Binários

```
        1 0 0 1   →  9₁₀
x       1 0 1 0   →  10₁₀
      ---------
        0 0 0 0
      1 0 0 1
    0 0 0 0
  1 0 0 1
  -------------
  1 0 1 1 0 1 0   →  90₁₀
```

$1001 \rightarrow 9_{10}$

$1010 \rightarrow 10_{10}$

$1011010 \rightarrow 90_{10}$

caso	resp
0x0	0
0x1	0
1x0	0
1x1	1

Multiplicação de Palavras

	0	0	0	0	1	0	0	1	9_{10}
x	0	0	0	0	1	0	1	0	10_{10}

a	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
b	0	0	0	0	0	0	0	0	0	0	0	1	0	0	1	0
c	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
d	0	0	0	0	0	0	0	0	0	1	0	0	1	0	0	0
e	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
f	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
g	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
h	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
	0	0	0	0	0	0	0	0	0	1	0	1	1	0	1	0

90_{10}

Somas com deslocamento para a Esquerda

	a	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	
	b	0	0	0	0	0	0	0	0	0	0	0	1	0	0	1	0	
i	'0'a+b'0'	0	0	0	0	0	0	0	0	0	0	0	1	0	0	1	0	18_{10}
	c	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	
j	'0'i+c'0'	0	0	0	0	0	0	0	0	0	0	0	1	0	0	1	0	18_{10}
	d	0	0	0	0	0	0	0	0	0	1	0	0	1	0	0	0	
k	'0'j+d'0'	0	0	0	0	0	0	0	0	0	1	0	1	1	0	1	0	90_{10}

⋮

Considerações acerca do MULT

- A multiplicação de dois números de 32 bits pode gerar potencialmente um número de 64 bits significativos;
- Todos os registradores possuem 32 bits;
- Utilizamos dois registradores especiais para armazenar o resultado de 32 bits de uma multiplicação;
- Dois registradores especiais:
  - HI → 32 bits mais significativos da palavra;
  - LO → 32 bits menos significativos da palavra.

Exemplo

⋮
long a,b,c = 42;
c = a*b;
⋮

a → $s0
b → $s1
c → $s2

```
mult $s0,$s1
mflo $s2
```

```
mul $s2,$s0,$s1
```

Instruções Aritméticas ($\mathbb{R}$)

OP	Descrição	Exemplo
abs.d	\$f2 ← |\$f4| valor absoluto (precisão dupla)	abs.d \$f2, \$f4
abs.s	\$f0 ← |\$f1| valor absoluto (precisão simples)	abs.d \$f0, \$f1
add.d	\$f2 ← \$f4 + \$f6 Soma em precisão dupla	add.d \$f2, \$f4, \$f6
add.s	\$f0 ← \$f1 + \$f2 Soma em precisão simples	add.d \$f0, \$f1, \$f2
c.eq.d	\$f2 == \$f4 ? CF0 ← 1 : CF0 ← 0	c.eq.d \$f2, \$f4
c.eq.s	\$f0 == \$f1 ? CF0 ← 1 : CF0 ← 0	c.eq.s \$f0, \$f1
c.le.d	\$f2 <= \$f4 ? CF0 ← 1 : CF0 ← 0	c.le.d \$f2, \$f4
c.le.s	\$f0 <= \$f1 ? CF0 ← 1 : CF0 ← 0	c.le.s \$f0, \$f1
c.lt.d	\$f2 < \$f4 ? CF0 ← 1 : CF0 ← 0	c.lt.d \$f2, \$f4
c.lt.s	\$f0 < \$f1 ? CF0 ← 1 : CF0 ← 0	c.lt.s \$f0, \$f1
ceil.w.d	Aproxima para cima para o inteiro (word)	ceil.w.d \$f1, \$f2

Instruções Aritméticas (ℝ)

OP	Descrição	Exemplo
ceil.w.s	Aproxima para cima para o inteiro (word)	ceil.w.s $f0, $f1
cvt.d.s	Converte de precisão simples para dupla	cvt.d.s $f2, $f1
cvt.d.w	Converte de inteiro 32 bits para precisão dupla	cvt.d.w $f2, $f1
cvt.s.d	Converte de precisão dupla para simples	cvt.s.d $f1, $f2
cvt.s.w	Converte inteiro 32bits para single	cvt.s.w $f0, $f1
cvt.w.d	Converte de precisão dupla para inteiro 32 bits	cvt.w.d $f1, $f2
cvt.w.s	Converte de precisão simples para inteiro 32 bits	cvt.w.s $f0, $f1
div.d	Divisão em ponto flutuante precisão dupla	div.d $f2, $f4, $f6
div.s	Divisão em ponto flutuante precisão simples	div.s $f0, $f1, $f3
floor.w.d	Arredonda para o inteiro 32 bits para baixo (double)	floor.w.d $f1, $f2
floor.w.s	Arredonda para o inteiro 32 bits para baixo (single)	floor.w.s $f0, $f1

Instruções Aritméticas (ℝ)

OP	Descrição	Exemplo
mov.d	Move o double em f4-f3 para f2-f3	mov.d $f2, $f4
mov.s	Move o single em f1 para f0	mov.s $f0, $f1
movf	Move t2 para t1 se CF0 = 0	movf $t1, $t2
movt	Move t2 para t1 se CF0 = 1	movt $t1,$t2
mul.d	Multiplicação em precisão dupla	mul.d $f2,$f4,$f6
mul.s	Multiplicação em precisão simples	mul.s $f0,$f1,$f3
neg.d	Negação double	neg.d $f2,$f4
neg.s	Negação single	neg.s $f0,$f1
sub.d	Subtração double	sub.d $f2,$f4,$f6
sub.s	Subtração single	sub.s $f0,$f1,$f3

Notação em Ponto Flutuante

- Fundamentada na notação numérica científica;

42,42 = $42{,}42 \times 10^0$ = $4{,}242 \times 10^1$ = $0{,}4242 \times 10^2$

- Utilização otimizada do espaço de representação;
- Note que o sinal fracionário "flutua" dependendo do expoente associado a base;

$$ {}^{+}\!/\!_{-}\, 0, mantissa \times base^{{}^{+}\!/\!_{-}\,\mathrm{exp}oente} $$

- A mantissa está contida no intervalo [0,1[
- É importante notar que a notação em ponto flutuante pode induzir à erros de arredondamento.

Padrões de Representação

IEEE Standard for Floating-Point Arithmetic, IEEE 754’2008

- Precisão Simples

- Precisão Dupla

Conversão (Precisão simples)

- Expoente possui um bias de 127 (01111111_2);
- Ao contrário da notação científica tradicional, que coloca todos os dígitos significativos a direita da vírgula, em ponto flutuante deixamos um ‘1’ a esquerda da vírgula.
- Equação para conversão binário →decimal:

$$n = (-1)^s \times \left(1 + \sum_{i=1}^{23} b_{23-i} \times 2^i \right) \times 2^{e-127}$$

Exemplo

- $10,25_{10}$ ➡ $1010,01_2$ ➡ $1,01001 \times 2^3$
- sinal → +
- expoente → 127+3 = 130 → (01111111+11) = 10000010
- mantissa → 01001000000000000000000

31	30 … 23	22 … 0
0	10000010	01001000000000000000000

Exemplo

$3DCCCCCD_H$

31	30 23	22 0
0	01111011	10011001100110011001101

- **sinal → +**
- **expoente→ 123-127= -4**

mantissa→ 1, 10011001100110011001101

$2^{0}+ 2^{-1} + 2^{-4} + 2^{-5} + 2^{-8} + 2^{-9} + 2^{-12} + 2^{-13} + 2^{-16} + 2^{-17} + 2^{-20} + 2^{-21} + 2^{-23} = 1.600000023841857900000000$

0.000160000002384185790000000

expoente	valor
2^{-1}	0,500000000000000
2^{-2}	0,250000000000000
2^{-3}	0,125000000000000
2^{-4}	0,062500000000000
2^{-5}	0,031250000000000
2^{-6}	0,015625000000000
2^{-7}	0,007812500000000
2^{-8}	0,003906250000000
2^{-9}	0,001953125000000
2^{-10}	0,000976562500000
2^{-11}	0,000488281250000
2^{-12}	0,000244140625000
2^{-13}	0,000122070312500
2^{-14}	6,10351562500000e-05
2^{-15}	3,05175781250000e-05
2^{-16}	1,52587890625000e-05
2^{-17}	7,62939453125000e-06
2^{-18}	3,81469726562500e-06
2^{-19}	1,90734863281250e-06
2^{-20}	9,53674316406250e-07
2^{-21}	4,76837158203125e-07
2^{-22}	2,38418579101563e-07
2^{-23}	1,19209289550781e-07
2^{-24}	5,96046447753906e-08
2^{-25}	2,98023223876953e-08

Casos Especiais

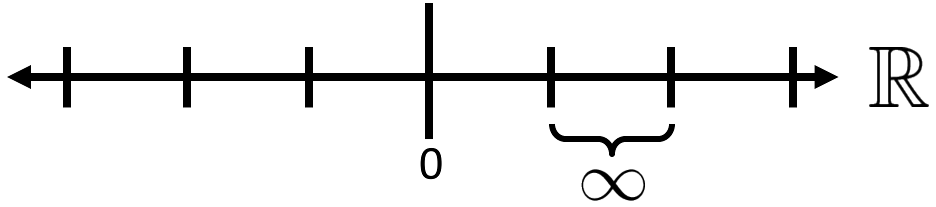
- Números (não normalizados)

	31	30 – 23	22 – 0
zero	**x**	**00000000**	**00000000000000000000000**
infinito	**x**	**11111111**	**00000000000000000000000**
NaN	**x**	**11111111**	**01000000010000000001000**

Pelo menos 1 bit da man-tissa diferente de zero

Números Representáveis

- Em matemática, o conjunto dos números reais é infinito;
- Entre dois números reais quaisquer, há infinitos números reais;
- Para tal, infinitos dígitos devem ser potencialmente utilizados;
- A representação de números reais utilizando a notação de ponto flutuante, utiliza um número finito de bits;
- Por definição, apenas **números racionais** podem ser representados em ponto flutuante;



Números Representáveis

- $0.1_{10} \rightarrow 0.0001100110011$ ....

$$Fra = \frac{1}{2^4} + \frac{1}{2^5} + \frac{1}{2^8} + \frac{1}{2^9} + \frac{1}{2^{12}} + \frac{1}{2^{13}} \cdots \rightarrow 0.1$$

- s = 0 | m = 1.1001100110011 ... e = -4

31	30 — 23	22 — 0
0	**01111100**	**10011001100110011001100**

- Convertendo de volta para decimal ...
- m = 0,100000001490116119384765625
- erro = 0,000000001490116119384765625

Exemplos

```
.data

pi:  .float  -3.14159
e:   .float  2.71828

.text
# Carregar dados da memória em registradores
#de ponto flutuante

       la        $t0, pi      #end. de pi em t0
       lwc1      $f0, 0($t0) #conteudo de pi em f0
       la        $t0, e     #end. de pi em t0
       lwc1      $f1, 0($t0) #conteudo de pi em f0

#valor absoluto de um número real
       abs.s     $f21, $f0
#imprimir um número real
       li        $v0, 2
       mov.s     $f12,$f21
       #syscall
#somar dois números reais
       add.s     $f12, $f0, $f1
       li        $v0, 2
       #syscall
#arredondamento
       ceil.w.s  $f12, $f21
       #cvt.w.s   $f12, $f12
       mfc1      $a0,  $f12
       li        $v0,  1
       syscall
```

Instruções Lógicas

OP	Descrição	Exemplo
and	E bit a bit	and $rd,$rs,$rt
or	OU bit a bit	or $rd,$rs,$rt
nor	NÃO OU bit a bit	nor $rd,$rs,$rt
xor	OU EXCLUSIVO bit a bit	xor $rd,$rs,$rt
andi	E bit a bit – segundo operando cte	andi $rt,$rs,cte
ori	OU bit a bit – segundo operando cte	ori $rt,$rs,cte
nori	NÃO OU bit a bit – segundo operando cte	nori $rt,$rs,cte
xori	OU EXCLUSIVO bit a bit – segundo operando cte	xori $rt,$rs,cte
ssl	Deslocamento para a esquerda (multiplicação base 2)	ssl $rt,$rs,[0:31]
slr	Deslocamento para a direita (divisão base 2)	slr $rt,$rs,[0:31]
sslv	Deslocamento para a esquerda (multiplicação base 2)	sslv $rd,$rs,$rt
slrv	Deslocamento para a direita (divisão base 2)	slrv $rd,$rs,$rt

Exemplo

- Testar se um número é par ou impar

```
addi $s0,$zero, 43          #coloca em s0 o num. a ser testado
addi $s1,$zero, 1           #máscara todos os bits 0 exceto o lsb
and  $t0,$s0,$s1            #zera todos os bits exceto o lsb

#se t0 for igual a 0 o número é par, se for igual a 1 é impar
```

- Operações lógicas também são utilizadas em diversos outros contextos:
  - Testes lógicos;
  - Computação gráfica;
  - Parsing de arquivos binários;
  - etc.

Multiplicação e Divisão via Deslocamento (Base 2)

0 0 0 0 0 0 1 1 ➔ 3_{10}

- ssl $s0, 1

0 0 0 0 0 1 1 0 ➔ 6_{10}

- Deslocar um número para a esquerda equivale a multiplica-lo por uma potência de 2;
- Deslocar um número para a direita equivale a dividi-lo por uma potência de 2

Bibliografia Comentada

- **PATTERSON, D. A. e HENNESSY, J. L. 2014.** *Organização e Projeto de Computadores – A Interface Hardware/Software.* Elsevier/ Campus 4ª edição.

- **HENNESSY, J. L. e PATTERSON, D. A. 2012.** *Arquitetura de Computadores – Uma Abordagem Quantitativa.* Elsevier/ Campus 5ª edição.

Bibliografia Comentada

- **MONTEIRO, M. A. 2001.** *Introdução à Organização de Computadores.* s.l. : LTC, 2001.

- **MURDOCCA, M. J. e HEURING, V. P. 2000.** *Introdução à Introdução de Computadores.* 2000. 85-352-0684-1.

Bibliografia Comentada

- **STALLINGS, W. 2002.** *Arquitetura e Organização de Computadores.* 2002.

- **TANENBAUM, A. S. 2007.** *Organização Estruturada de Computadores.* 2007.