

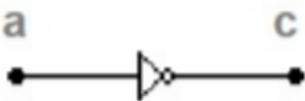
# NOTEBOOK

---

---

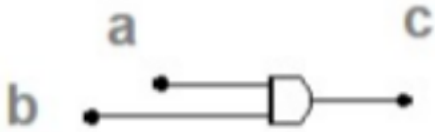
# BLOCOS CONTRUTIVOS

NOT (Inversora)  
 $c = \overline{a}$



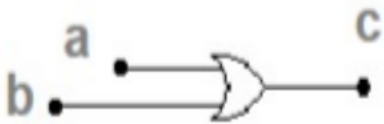
Entrada	Saída
<b>a</b>	<b>c</b>
0	1
1	0

AND  
 $c = a.b$



Entradas		Saída
<b>a</b>	<b>b</b>	<b>c</b>
0	0	0
0	1	0
1	0	0
1	1	1

OR  
 $c = a + b$



Entrada		Saída
<b>a</b>	<b>b</b>	<b>c</b>
0	0	0
0	1	1
1	0	1
1	1	1

# SOMADOR DE 1 BIT

SOMA: A + B + "Vai 1", E GERA O RESULTADO DO "Vai 1".

Entradas			Saídas		Comentários
A	B	Vem 1	Soma	Vai 1	
0	0	0	0	0	0+0+0 = 00
0	0	1	1	0	0+0+1 = 01
0	1	0	1	0	0+1+0 = 01
0	1	1	0	1	0+1+1 = 10
1	0	0	1	0	1+0+0 = 01
1	0	1	0	1	1+0+1 = 10
1	1	0	0	1	1+1+0 = 10
1	1	1	1	1	1+1+1 = 11



# MEIO SOMADOR

FAZENDO A SOMA BINÁRIA, VOCÊ TEM 2 OPÇÕES DE SAÍDA (0, 1). CASO A SOMA SEJA 1+1 OCORRE O ESTOURO. ENTÃO USAMOS A TÉCNICA ACIMA COMO DEMONSTRADO.

# Binário

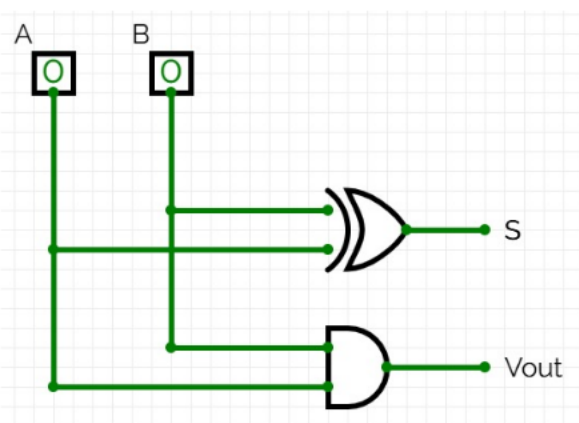
Diagram illustrating a 1-bit adder circuit. The inputs are two 4-bit numbers, 11101 and 01101, which are added to produce the output 101010. The carry bits (1, 1, 1) are shown above the first three columns of the addition. The final carry-out (1) is shown above the fifth column. The inputs are labeled "Entrada A" and "Entrada B", and the output is labeled "Saída S". The circuit is labeled "Circuito Somador de um bit".

ENTRADA A	ENTRADA B	SÁIDA S	SÁIDA VAI UM
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1

Diagram illustrating the logic for the carry-out (Vai Um) in a full adder. The inputs are A and B, and the output is S (Sum) and V<sub>out</sub> (Carry-out). The truth table shows that V<sub>out</sub> is 1 only when both A and B are 1, which is the output of an AND gate.

SÓ OBTEMOS 1 NA SAÍDA QUANDO AS ENTRADAS SÃO DIFERENTES. A PORTA LÓGICA XOR REPRESENTA A SAÍDA S DE ACORDO COM A TABELA

SÓ OBTAMOS 1 NA VOUT  
QUANDO AS ENTRADAS SÃO 1.  
LOGO A PORTA LÓGICA AND  
REPRESENTA A SAÍDA VOUT



DIANTE DISTO, TEMOS O CIRCUITO DO MEIO SOMADOR  
CASO A ENTRADA A FOR  
DIFERENTE DA ENTRADA B, A  
PORTA XOR SERÁ ATIVA, E CASO  
AS PORTAS A E B FOREM 1, A  
PORTA AND SERÁ ATIVA

## MODELO MATEMÁTICOS

$$\begin{array}{r}
 1001 \\
 1011 \\
 1001 \\
 \hline
 10100
 \end{array}$$

M BITS + M BITS = M BITS

$$\begin{array}{cccc} 0 & 1 & 1 & 0 \\ 0 & 0 & 1 & 1 \\ \hline 0 & 1 & 1 & 0 \end{array}^*$$


$$\begin{matrix} \text{M} & * & \text{M} & = & \text{M} + \text{M} \\ \text{BITS} & & \text{BITS} & & \end{matrix}$$

$$\begin{array}{ccccccc} & & & 0 & 1 & 1 & 0 \\ & & 0 & 0 & 0 & 0 & \\ & 0 & 0 & 0 & 0 & & \\ \hline 0 & 0 & 1 & 0 & 0 & 1 & 0 \end{array}$$

# CONVERSÃO DE BASES PARA BINÁRIO

•DA BASE 10 PARA BINARIO, FAZEMOS O MÉTODO DA DIVISÃO POR 2.


Ex: 1984

1984	% 2 = 992,	RESTO 0		LENDO OS NÚMEROS DE BAIXO PARADIGMA TEMOS O NÚMERO CONVERTIDO DA BASE 10 PARA BINARIO
992	% 2 = 496,	RESTO 0		
496	% 2 = 248,	RESTO 0		
248	% 2 = 124,	RESTO 0		
124	% 2 = 62,	RESTO 0		
62	% 2 = 31,	RESTO 0		
31	% 2 =15,	RESTO 1		
15	% 2 =7,	RESTO 1		
7	% 2 =3,	RESTO 1		
3	% 2 =1,	RESTO 1		
1	% 2 =0,	RESTO 1		

RESPOSTA: 1111100000

•DA BASE 16 PARA BINÁRIO, PEGAMOS CADA CARÁCTER SEPARADO E CONVERTEMOS.

Ex: 4B0

4 --> 0100		CONVERTO CADA CARACTERE SEPARADAMENTE E LOGO APÓS JUNTO TODOS.
B (11) --> 1011		
0 --> 0000		

RESPOSTA: 0100 1011 0000

•DA BASE 8 PARA 16, CONVERTEMOS O NÚMERO PARA A BASE 10 E COM O RESULTADO PASSAMOS PARA A BASE 16.

Ex: 806

806 NA BASE 8 EQUIVALE A:  $8 * 8^2 + 0 * 8^1 + 6 * 8^0 = 518$

OU SEJA, 806 (BASE 8) É 518 NA BASE 10.

PARA TRANSFORMARMOS O NÚMERO PARA BASE 16, VAMOS DIVIDINDO POR 16.

$518 \div 16 = 32$ , COM O RESTO 6  
 $32 \div 16 = 2$ , COM O RESTO 0  
 $2 \div 16 = 0$ , COM O RESTO 2

PORTANTO 518 NA BASE 10 SE TORNA 206 NA BASE HEXADECIMAL.

ASSIM CHEGAMOS A CONCLUSÃO QUE 806 NA BASE 8 E EQUIVALENTE A 206 NA BASE HEXADECIMAL.

## LIMITES DE DÍGITOS POR BASE

BASE 2 (BINARIA ): NÚMEROS COMPOSTOS POR 0 OU 1.

BASE 8 (OCTAL): NÚMEROS COMPOSTOS DE 0 A 7.

BASE 10 (DECIMAL): NÚMEROS COMPOSTOS DE 0 A 9.

BASE 16 (HEXA):NÚMEROS COMPOSTOS DE 0 A 9 E LETRAS DE A A F.

# MIPS

MIPS É UMA LINGUAGEM COMPUTACIONAL.

EM ASSEMBLY NÃO USAMOS VARIÁVEIS, USAMOS REGISTRADORES. E DENTRO DO MIPS EXISTEM 32 REGISTRADORES.

CADA REGISTRADOR TEM 32 BITS, E SÃO NUMERADOS DE 0 A 31. POR CONVENÇÃO CADA REGISTRADOR TEM UM NOME PARA FACILITAR A CODIFICAÇÃO, DESTES NOMES SE INICIAM COM \$.

MIP	LINGUAGEM C
ADD A, B, C	A = B + C
SUB D, E, F	D = E - F

## EXEMPLO 1

ADIÇÃO EM ASSEMBLY EM COMPARAÇÃO COM C:

EM C:

INT F, G, H;

F = G + H;

EM MIPS:

# INÍCIO  
F ---> \$S0  
G ---> \$S1  
H ---> \$S2

ADD \$S0, \$S1, \$S2 # F = G + H  
#FIM

# EM MIPS É O MESMO QUE // EM C (COMENTÁRIO)

## EXEMPLO 2

ADIÇÃO EM ASSEMBLY EM COMPARAÇÃO COM C:

EM C:

INT F, G, H, I, J;

F = (G + H) - (I + J);

EM MIPS:

# INÍCIO  
F ---> \$S0  
G ---> \$S1  
H ---> \$S2  
I ---> \$S3  
J ---> \$S4  
\$T0  
\$T1

ADD \$T0, \$S1, \$S2 # T0 = G + H  
ADD \$T1, \$S3, \$S4 # T1 = I + J  
SUB \$S0, \$T0, \$T1 # F = T0 - T1  
#FIM

## IMEDIATOS

IMEDIATOS: SÃO CONSTANTES NUMÉRICAS.

ADDI \$s0, \$s1, 10 # s0 = s1 + 10

### PORQUE ADDI E NÃO ADD?

ADD SERVE PARA SOMAR APENAS REGISTRADORES E ADDI SERVE PARA ADICIONAR IMEDIATOS AOS REGISTRADORES.

ADD = REGISTRADORES + REGISTRADORES  
ADDI = REGISTRADORES + IMEDIATOS

## EXERCÍCIO 1:

LINGUAGEM C:

```
A = 10;  
B = -1;  
A = 4 * A + 1;  
C = A + B;
```

MIPS:

```
A ---> $S0  
B ---> $S1  
C ---> $S2
```

```
ADDI $S0, $S0, 10    # A = A + 10  
ADDI $S1, $S1, -1    # B = B - 1  
LI $T0, 4             # COLOCA O NÚMERO 4 NO REGIS $T0  
MULT $S0, $S0, $T0    # A = A * 4  
ADDI $S0, $S0, 1      # A = A + 1  
ADD $S3, $S0, $S1     # C = A + B
```

UM IMEDIATO PARTICULAR , O NÚMERO ZERO (0) APARECE MUITO FREQUENTEMENTE EM CÓDIGO.

ENTÃO NÓS DEFINIMOS O REGISTRADOR ZERO ( \$0 \$ZERO ) PARA SEMPRE TER O VALOR 0.

## OPERADORES LÓGICOS

AND - SAÍDA 1 SOMENTE SE AMBAS AS ENTRADAS FOREM 1.

OR - SAÍDA 1 SE PELO MENOS UMA ENTRADA FOR 1

AND, OR : AMBAS ESPERAM UM TERCEIRO ARGUMENTO SER UM REGISTRADOR.

```
AND $T0, $S1, $S2    # T0 = s1 & s2  
OR $T0, $S1, $S2     # T0 = s1 || s2
```

ANDI, ORI : AMBAS ESPERAM UM TERCEIRO ARGUMENTO SER UM IMEDIATO.

```
ANDI $T0, $S1, 3     # T0 = s1 & 3  
ORI $T0, $S1, 3      # T0 = s1 || 3
```

## SHIFT

MOVE TODOS OS BITS NA PALAVRA PARA ESQUERDA OU DIREITA UM CERTO NÚMERO DE BITS.

**EXEMPLO: SHIFT RIGTH POR 8 BITS**

```
0001 0010 0011 0100 0101 0110 0111 1000  
      ↘                ↘  
0000 0000 0001 0010 0011 0100 0101 0110
```

**EXEMPLO: SHIFT LEFT POR 8 BITS**

```
001 0010 0011 0100 0101 0110 0111 1000  
      ↙                ↙  
0011 0100 0101 0110 0111 1000 0000 0000
```

# SINTAXE DA INSTRUÇÃO SHIFT

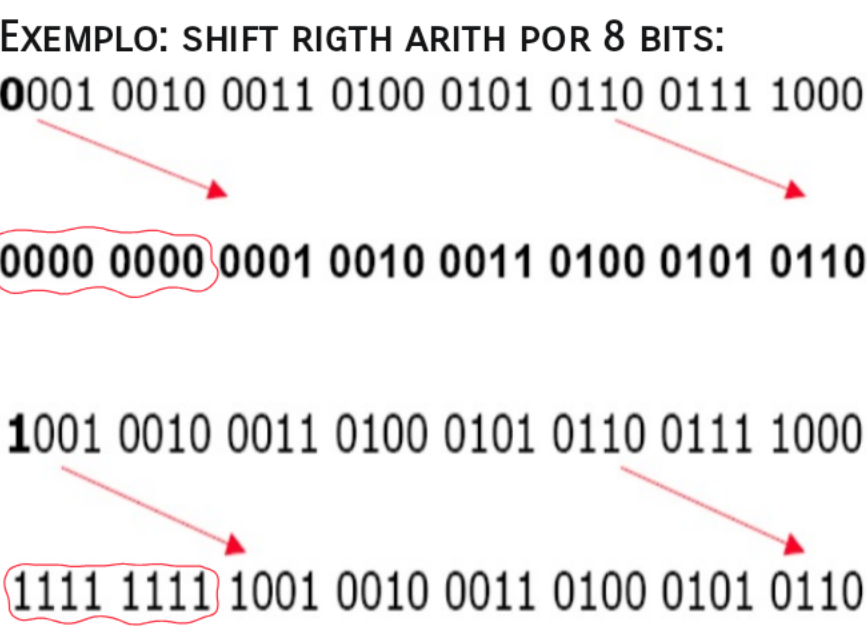
- 1- OPERAÇÃO
- 2- REGISTRADOR QUE RECEBERÁ O VALOR
- 3- REGISTRADOR QUE SERÁ DESLOCADO
- 4- QUANTIDADE DE DESLOCAMENTO ( <= 32)

EXEMPLO: SLL \$S1, \$S2, 8      # S1 = S2 << 8

SLL (SHIFT LEFT LOGICAL): DESLOCA PARA ESQUERDA E COMPLETA OS BITS ESVAZIANDO COM 0s.

SRL (SHIFT RIGTH LOGICAL): DESLOCA PARA DIREITA E COMPLETA OS BITS ESVAZIANDO COM 0s.

SRA (SHIFT RIGTH ARITHMETIC): DESLOCA PARA DIREITA E PREENCHE OS BITS ESVAZIADOS ESTENDENDO O SINAL.



## EM DECIMAL:

MULTIPLICAR POR 10 E O MESMO QUE DESLOCAR PARA A ESQUERDA POR 1

714 x 10 = 7140

56 x 10 = 560

MULTIPLICAR POR 100 E O MESMO QUE DESLOCAR PARA A ESQUERDA POR 2

714 x 100 = 71400

56 x 100 = 5600

MULTIPLICAR POR 10^N E O MESMO QUE DESLOCAR PARA A ESQUERDA N VEZES.

## EM BINARIO:

MULTIPLICAR POR 2 E O MESMO QUE DESLOCAR PARA A ESQUERDA POR 1

11 x 10 = 110

1010 x 10 = 10100

MULTIPLICAR POR 4 E O MESMO QUE DESLOCAR PARA A ESQUERDA POR 2

11 x 100 = 1100

1010 x 100 = 101000

MULTIPLICAR POR 2^N E O MESMO QUE DESLOCAR PARA A ESQUERDA N VEZES.

Instrução	O que faz	Preenche com	Usa para
sll	Desloca bits para a esquerda	0	Multiplicação por 2^n
srl	Desloca bits para a direita	0	Divisão sem sinal
sra	Desloca bits para a direita	Bit de sinal	Divisão com sinal preservado



EM C MULTIPLICAMOS POR UMA POTÊNCIA DE 2 E VEMOS A COMPARAÇÃO DA INSTRUÇÃO SHIFT A ELE:

A \*= 8 ; (EM C)

SERIA COMPILADO COMO:

sll \$s0, \$s0, 3 ( EM MIPS)

\*DA MESMA FORMA DESLOQUE PARA A DIREITA PARA DIVIDIR POR POTÊNCIA DE 2.

## MEMÓRIA

INSTRUÇÕES PARA TRANSFERÊNCIA DE DADOS. TRANSFEREM DADOS ENTRE REGISTRADORES E A MEMÓRIA

- MEMÓRIA PARA O REGISTRADOR -----> LOAD
- REGISTRADOR PARA A MEMÓRIA -----> STORE

### SINTAXE DA INSTRUÇÃO LOAD

- 1- NOME DA OPERAÇÃO (LW)
- 2- REGISTRADOR QUE RECEBERÁ O VALOR
- 3- DESLOCAMENTO NUMÉRICO EM BYTES
- 4- REGISTRADOR CONTENDO O PONTEIRO PARA A MEMÓRIA

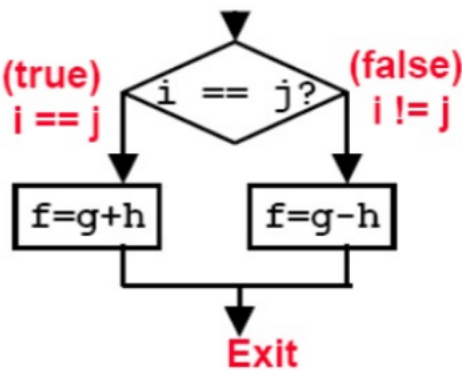
EXEMPLO: LW \$T0, 12 (\$s0)

\$s0 É CHAMADO DE REGISTRADOR BASE  
12 É A QUANTIDADE DE DESLOCAMENTO (OFFSET)

\*SINTAXE DA INSTRUÇÃO STORE É MESMA QUE LOAD, PORÉM AO INVÉS DE LW, É SW.

## IF, IF-ELSE

EM C É MUITO HABITUAL USAR IF ELSE, AGORA VAMOS VER COM ESSE EXEMPLO COMO FAZEMOS EM MIPS.

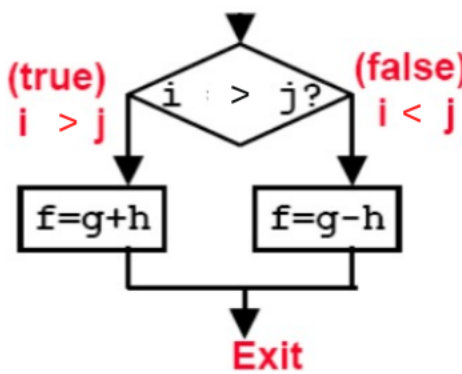


```
BEQ $T0, $T1 TRUE # SE X == Y, VA PARA TRUE
SUB $s1, $s2, $s3 # (FALSO) ELSE F = G - H

J FIM # USADO PARA PULAR O IF

TRUE: ADD $s1, $s2, $s3 # (TRUE) IF F = G + H

FIM: # FIM DA CONDIÇÃO
```



```
BLE $T0, $T1 TRUE # SE I > J, VÁ PARA TRUE
SUB $s1, $s2, $s3 # (FALSO) ELSE F = G - H

J FIM # USADO PARA PULAR O IF

TRUE: ADD $s1, $s2, $s3 # (TRUE) IF F = G + H

FIM: # FIM DA CONDIÇÃO
```

- `beq $s1, $s2, label` – if `s1 == s2`
- `bne $s1, $s2, label` – if `s1 != s2`
- `blt $s1, $s2, label` – if `s1 < s2` (*pseudo-instrução*)
- `bgt $s1, $s2, label` – if `s1 > s2` (*pseudo-instrução*)
- `ble $s1, $s2, label` – if `s1 <= s2` (*pseudo-instrução*)
- `bge $s1, $s2, label` – if `s1 >= s2` (*pseudo-instrução*)



DO WHILE