

Arquitetura de Computadores

Prof. Fábio Eduardo Vieira Angelo
UNISUL – 2019

Material baseado nos Slides da Professora Sheila Santisi Travessa

Aritmética Computacional

- As palavras em um computador são armazenadas na forma de números binários. Veremos como:
 - Representar números negativos.
 - Qual o maior número que poderemos representar em uma palavra de computador.
 - O que acontece se uma determinada operação gera um número maior que o que pode ser manipulado pelo computador.

Números em um Computador

- Os números podem ser representados em diferentes bases, mas daremos atenção apenas aos sistemas binário e decimal.
- Da aritmética sabemos que:

$$d.base^i$$

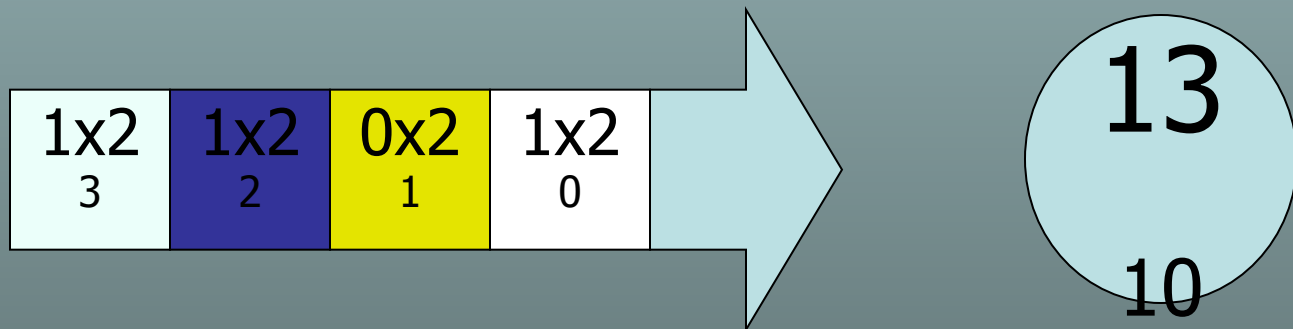
- Isto nos leva a uma forma óbvia de contar:

$$1.10^3 + 1.10^2 + 2.10^1 + 6.10^0 = 1126$$

| | | | | | | |
|--------|--------|--------|---|-----------|-----------|-----------|
| 10^2 | 10^1 | 10^0 | , | 10^{-1} | 10^{-2} | 10^{-3} |
|--------|--------|--------|---|-----------|-----------|-----------|

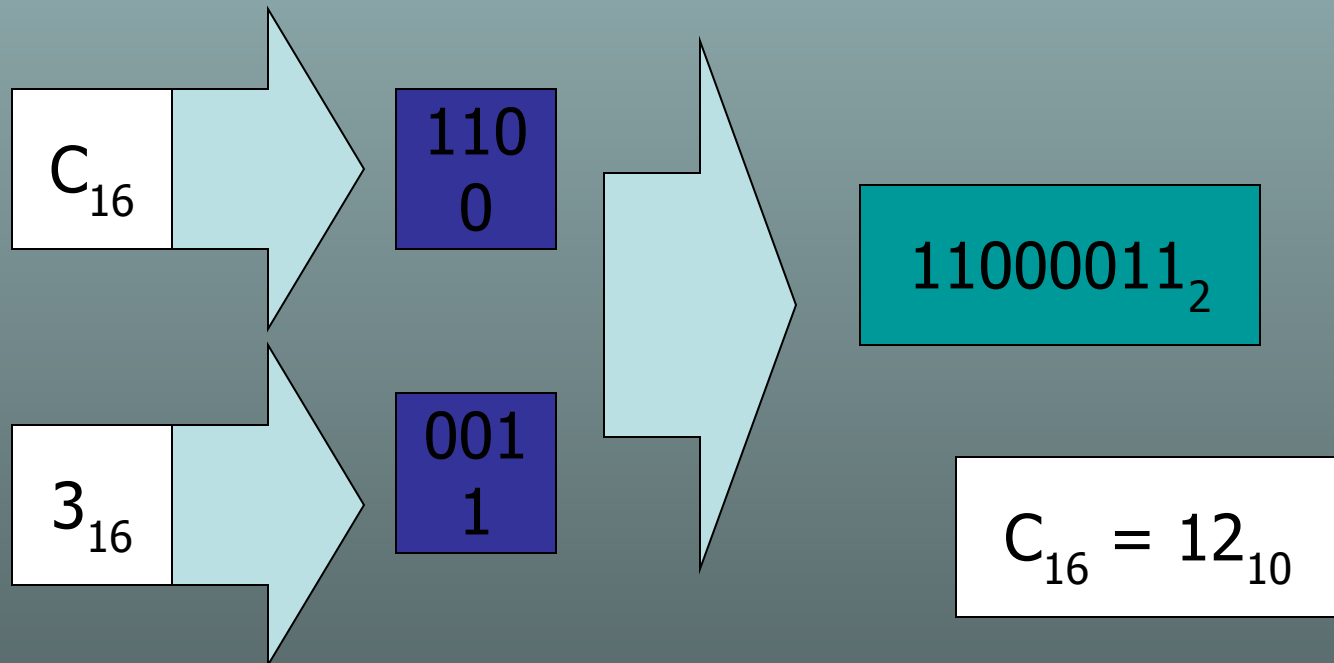
Números em um Computador

- No sistema binário funciona da mesma maneira:



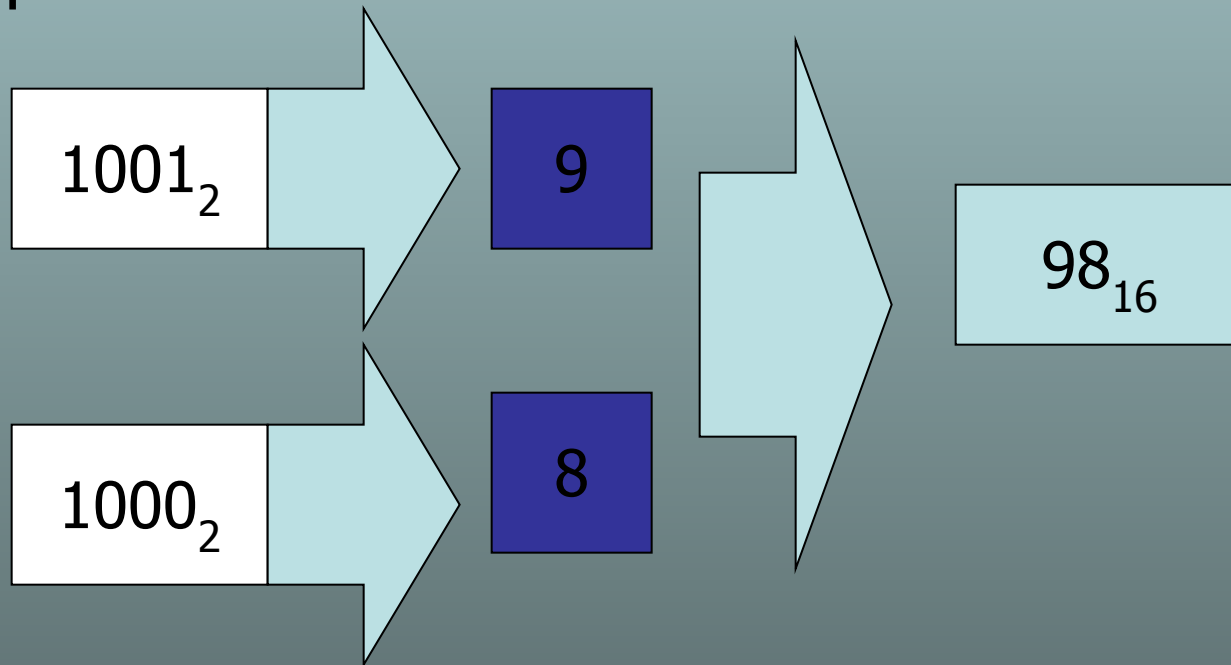
Conversão Hexa \square Binário

- É feita através da conversão de cada algarismo hexa para o binário utilizando sempre quatro bits.



Conversão Binário ☐ Hexa

- É feita agrupando-se os bits quatro a quatro e convertendo para o hexadecimal correspondente.



Números em um Computador

- O esquema a seguir mostra a numeração dos bits em uma palavra do MIPS.

| | | | | | | | |
|-----------|-----------|-----------|-----------|-----------|------|------|------|
| 31-2 8 | 27-2 4 | 23-2 0 | 19-1 6 | 15-1 2 | 11-8 | 7-4 | 3-0 |
| 0000 | 0000 | 0000 | 0000 | 0000 | 0000 | 0000 | 0000 |

- O bit menos significativo é chamado LSB.
- O bit mais significativo é chamado MSB.

Números em um Computador

- As palavras do MIPS têm 32 bits, portanto podemos representar nela 2^{32} padrões diferentes de 32 bits.
 - Tais combinações representam os números decimais inteiros de 0 a $2^{32}-1$ ($4.294.697.295_{10}$).

| | | | | | | | | |
|------|------|------|------|------|------|------|------|----------|
| 0000 | 0000 | 0000 | 0000 | 0000 | 0000 | 0000 | 0000 | 0_{10} |
| 0000 | 0000 | 0000 | 0000 | 0000 | 0000 | 0000 | 0001 | 1_{10} |

| | | | | | | | | |
|------|------|------|------|------|------|------|------|----------------------|
| 1111 | 1111 | 1111 | 1111 | 1111 | 1111 | 1111 | 1110 | $4.294.697.294_{10}$ |
| 1111 | 1111 | 1111 | 1111 | 1111 | 1111 | 1111 | 1111 | $4.294.697.295_{10}$ |

Números em um Computador

- Porque não trabalhar com a base 10?
 - O primeiro computador comercial trabalhava com aritmética decimal, operando valores expressos em um código de caracteres.
 - Entretanto, tal código era montado como um string binário.
 - Tal técnica mostrou-se tão ineficiente que todas as máquinas subseqüentes a abandonaram.

Números em um Computador

- É possível representar números como strings de caracteres ASCII, mas o hardware necessário para operar tais números é muito mais complexo do que o necessário para a utilização de binário (complemento a 2).
 - Por exemplo, para representar um bilhão na base 10, utiliza-se 10 dígitos. (1.000.000.000)
 - O MIPS é capaz de representa-lo dentro de seus 32 bits.(0011 1011 1001 1010 1100 1010 0000 0000)
 - Em ASCII cada dígito consome 8 bits. Dessa forma, este número necessitaria de \square $(10 \times 8) = 80$ bits.
 - Ou seja, a representação em ASCII consumiria 2,5 vezes mais bits.

Números com Sinal

- Os programas tratam de números negativos e positivos, por isso é necessário que se utilize uma representação que os distinga.
- A solução mais óbvia e já tentada foi a representação sinal magnitude.

Números com Sinal

- Apesar de sua aparente conveniência, a representação sinal/magnitude tem alguns defeitos:
 - Não fica claro onde se deve colocar o bit de sinal.
 - O uso do bit separado para o sinal provoca o surgimento do zero positivo e zero negativo, o que pode causar problemas de programação.

Números com Sinal

- Devido aos problemas apresentados por todas as tentativas de representação de números negativos, os computadores atuais utilizam a convenção conhecida como complemento a 2, complemento de 2 ou simplesmente complemento 2.
 - A representação em complemento a 2 tem a vantagem de representar números negativos com o bit mais significativo em 1. Portanto, o hardware só precisa testar o primeiro bit para ver se o número é positivo ou negativo.
 - Neste caso o bit mais significativo costuma ser chamado de bit de sinal.

Formato Complemento a 1

- É obtido através da troca do 0's por 1 e dos 1's por 0.
 - **101101** ☐ **número original**
 - **010010** ☐ **complementos a 1**

Formato Complemento a 2

- É obtido tomando-se o complemento a 1 do número, e adicionando-se uma unidade ao seu bit menos significativo.
 - **101101(45) □ 010010 □ 010011**
 - **101100(44) □ 010011 □ 010100**

Tabela representativa da sequência de números binários positivos e negativos

| Decimal | -9 | -8 | -7 | -6 | -5 | -4 | -3 | -2 | -1 |
|------------------|-------|-------|-------|-------|-------|-------|-------|-------|-------|
| Binário | -1001 | -1000 | -0111 | -0110 | -0101 | -0100 | -0011 | -0010 | -0001 |
| Complemento 2 | 0111 | 1000 | 1001 | 1010 | 1011 | 1100 | 1101 | 1110 | 1111 |

| Decimal | 0 | +1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |
|-------------------|-------|-------|-------|-------|-------|-------|-------|-------|-------|-------|
| Binário | +0000 | +0001 | +0010 | +0011 | +0100 | +0101 | +0110 | +0111 | +1000 | +1001 |
| Complem ento 2 | 0000 | 0001 | 0010 | 0011 | 0100 | 0101 | 0110 | 0111 | 1000 | 1001 |

Números com Sinal em Complemento a 2

- Se o número é positivo, a magnitude é representada na forma binária normal, com um bit de sinal de valor 0.

| | | | | | | | |
|---|---|---|---|---|---|---|------|
| 0 | 1 | 0 | 1 | 1 | 0 | 1 | =+45 |
|---|---|---|---|---|---|---|------|

- Se o número é negativo, a magnitude é representada na forma complemento a 2 e um bit de valor 1 é colocada à esquerda do bit mais significativo.

| | | | | | | | |
|---|---|---|---|---|---|---|------|
| 1 | 0 | 1 | 0 | 0 | 1 | 1 | =-45 |
|---|---|---|---|---|---|---|------|

Números com Sinal em Complemento a 2

- O sistema em complemento de 2 é usado para representar números com sinal porque permite transformar a operação de subtração em uma adição, o que significa economia e simplificação de hardware.

Adição em Complemento a 2

- Os sistema em complemento a 1 e a 2 são bastante similares, mas o último é mais utilizado por facilidade de implementação.
 - **Dois Números Positivos (+9 e +4)**

| | | |
|----|---|------|
| +9 | 0 | 1001 |
| +4 | 0 | 0100 |
| | 0 | 1101 |

Adição em Complemento a 2

- Um número positivo e um negativo de valor absoluto menor

| | | |
|----|---|------|
| +9 | 0 | 1001 |
| -4 | 1 | 1100 |
| | 0 | 0101 |

- Um número positivo e um negativo de valor absoluto maior

| | | |
|----|---|------|
| -9 | 1 | 0111 |
| +4 | 0 | 0100 |
| | 1 | 1011 |

Adição em Complemento a 2

– Dois valores negativos

| | | |
|----|---|------|
| -9 | 1 | 0111 |
| -4 | 1 | 1100 |
| | 1 | 0011 |

- Como se pode observar, a manipulação e a identificação de valores negativos em adições fica extremamente facilitada com a convenção complemento a 2.

Operações aritméticas no sistema binário

- Adição no sistema binário:
 - Devemos agir como numa adição convencional no sistema decimal, lembrando que, no sistema binário temos apenas 2 algarismos.

$$11001 + 1011 = 100100$$

Subtração no sistema binário

- O método de resolução é análogo a uma subtração no sistema decimal:

$$111 - 100 = 011$$

Multiplicação no sistema binário

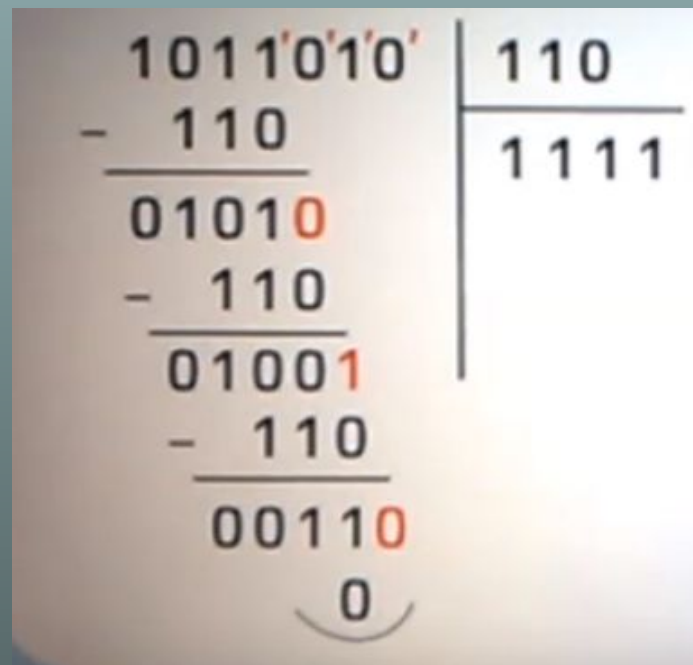
- Procede-se como uma multiplicação no sistema decimal:

$$11011 \times 10 = 110110$$

Divisão no sistema binário

- Procede-se como uma divisão no sistema decimal:

$$1011010 / 110 = 1111$$



Handwritten binary division showing the steps:

$$\begin{array}{r} 1011010' \\ - 110 \\ \hline 01010 \\ - 110 \\ \hline 01001 \\ - 110 \\ \hline 00110 \\ \hline 0 \end{array}$$

The quotient is 1111.

Subtração em Complemento a 2

- Para subtrair um número binário (subtraendo) de outro (minuendo), deve-se:
 - Tomar o complemento a dois do subtraendo, incluindo o bit de sinal (inversão de sinal).
 - Adicionar o minuendo ao subtraendo.

Subtração em Complemento a 2

- Tomemos como exemplo o caso em que +4 deve ser subtraído de +9.

– Minuendo (+9) \square 01001

– Subtraendo (+4) \square 00100

$$\begin{array}{r} 01001 (+9) \\ +11100 (-4) \\ \hline 00101 (+5) \end{array}$$

Números Com Sinal

- Como foi visto, a necessidade de se trabalhar com números positivos e negativos em uma palavra de computador e considerando-se os prós e contras das opções já testadas, fez com que a escolha recaísse sobre a representação em complemento a 2.
 - Desde 1965 esta representação é usada em praticamente todos os computadores construídos.

Números Com Sinal

- Caso uma adição ou uma subtração gere um resultado que não possa ser armazenado no hardware disponível (no caso do MIPS 32 bits), diz-se que houve um overflow.

Overflow

- Os erros de overflow podem ocorrer quando se executam adições, subtrações, multiplicações e acesso à memória.
- Para operações em que se representa endereços de memória, utiliza-se números sem sinal em que o overflow é ignorado.
 - Os projetistas de máquinas devem, portanto, ter um meio de ignorar o overflow em alguns casos e considerar em outros.
 - No caso do MIPS a solução adotada consiste da utilização de instruções aritméticas diferentes para cada caso:
 - Soma(add) e subtração (sub) □ sinalizam o overflow.
 - Soma sem sinal (addu) e subtração sem sinal (subu) □ não sinalizam o overflow.

Overflow

- O projetista da máquina precisa decidir como tratar a ocorrência do overflow.
 - Algumas linguagens como o C deixam o tratamento por conta do hardware.
 - Pode existir a detecção por hardware que gera uma exceção, e que depois é tratada por software.

Overflow

- No caso do MIPS o overflow é detectado por meio de uma interrupção (chamada a um procedimento não programada).
 - O endereço da instrução que causou a interrupção é armazenado em um registrador, o que permite que o programa seja retomado depois da execução de uma rotina corretiva.

Interface Hardware x Software

- Os endereços de memória são sempre positivos, portanto:
 - Em algumas situações os programas precisam tratar números positivos e negativos.
 - Em outra precisam tratar somente positivos.
- As linguagens de programação costumam mostrar tal distinção. Por exemplo, em C:
 - Sem sinal □ unsigned int.
 - Com sinal □ int
- Baseado nisto, deve-se lembrar que um número iniciado por um bit 1, representa grandezas diferentes dependendo da abordagem.

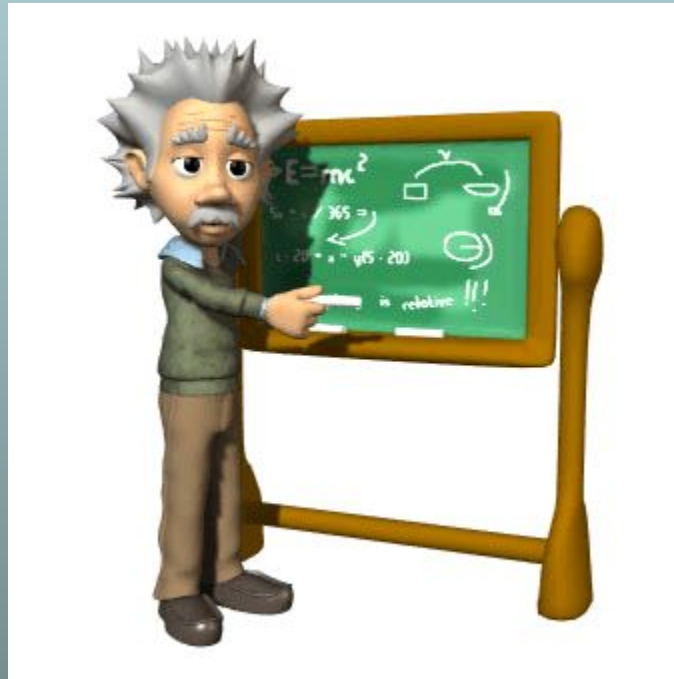
Operações Lógicas

- Apesar de os primeiros computadores terem concentrado suas operações em palavras, ficou logo claro que seria muito útil operar em campos de bits, ou bits individuais.
 - Com base nisto, algumas instruções foram adicionadas à arquitetura das máquinas, com o intuito de, entre outras coisas, facilitar a colocação e extração de bits dentro de palavras.
- Uma dessas operações é o “shift”, que move todos os bits de uma palavra para a esquerda ou para a direita, preenchendo com 0 os bits vazios.
 - Para que serve deslocar à direita e à esquerda?

Operações Lógicas

- Existem operações lógicas que permitem um deslocamento de oito bits.
- O nome das instruções de deslocamento do MIPS são sll (shift left logical) e srl (shift right logical).
- A instrução para deslocar o conteúdo do registrador \$s0 oito bits à esquerda e armazená-lo em \$t2 é:
 - sll \$t2,\$s0,8
 - O campo shamt (shift amount) da instrução formato R armazena a quantidade de bits a ser deslocada.

Operações Lógicas



Deslocamento pode ser usado para Multiplicação ou Divisão!

Operações Lógicas

- Outras operações lógicas muito utilizadas são AND e OR.
 - Na função E, a saída tem nível lógico 1 somente quando as duas entradas tem nível lógico 1.
 - Na função OU, a saída terá nível lógico 1 sempre que uma das entradas for 1.
 - Na função Ou exclusivo a saída é 1 quando apenas uma das entradas for 1.

Operações Lógicas

- A operação AND no MIPS é feita por:
 - and \$t0,\$t1,\$t2
- A operação OR no MIPS é feita por:
 - or \$t0,\$t1,\$t2

Interface Hardware/Software

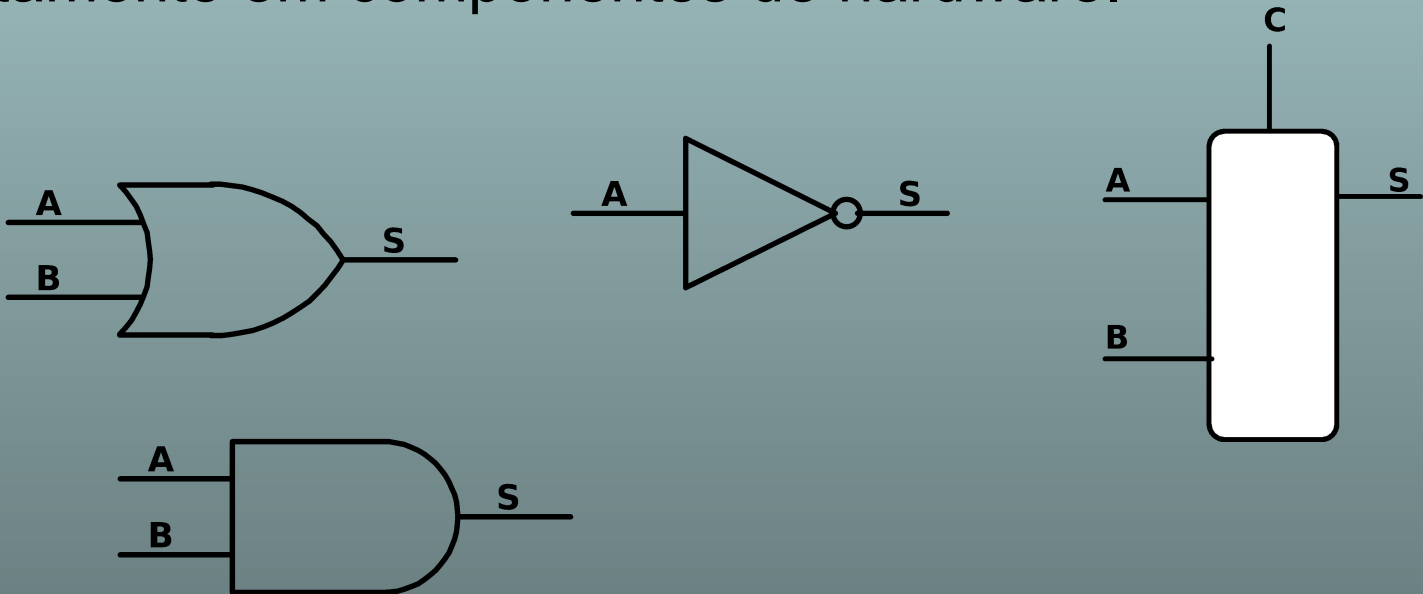
- A linguagem C permite que campos de bits sejam definidos dentro de uma palavra, permitindo que tanto que objetos sejam inseridos na palavra quanto que tais objetos correspondam ao valor colocado por uma interface externa, como um dispositivo de E/S.

Unidade Lógica Aritmética

- É o dispositivo que realiza as operações lógicas como AND e OR e aritméticas como adição e subtração em um processador.
- É baseada em blocos lógicos básicos como AND, OR, Inversor e MUX.
- Como a palavra do MIPS tem 32 bits, necessita de uma ULA de 32 bits.
- Iniciaremos com a análise de uma ULA de 1 bit.

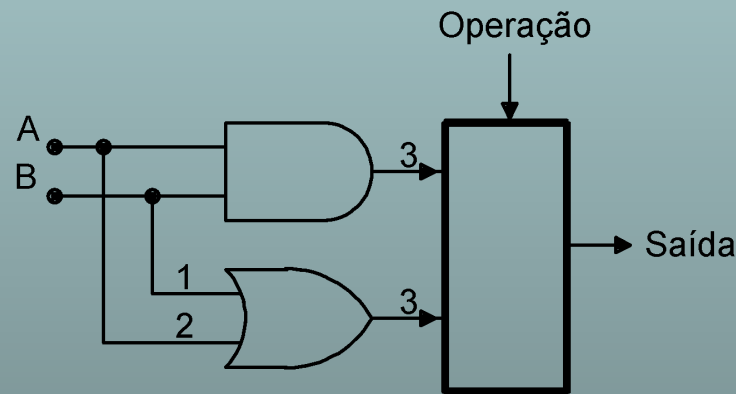
Unidade Lógica Aritmética

- As implementações das operações lógicas são as mais simples de serem realizadas, pois são mapeadas diretamente em componentes de hardware.



ULA de 1 Bit

- Unidade Lógica de 1 bit.



- Deve-se agora agregar o circuito capaz de executar a soma.

ULA de 1 Bit

- A adição de número binários é realizada da mesma forma que a adição de números decimais.
 - O dígito menos significativo é o primeiro a ser operado.
 - As possibilidades existentes são:

$$0 + 0 = 0$$

$$0 + 1 = 1$$

$$1 + 1 = 0 \rightarrow \text{carry 1 para a posição seguinte}$$

$$1 + 1 + 1 = 1 \rightarrow \text{carry 1 para a posição seguinte}$$

- Os circuitos que executam tal função são o meio-somador e o somador completo,

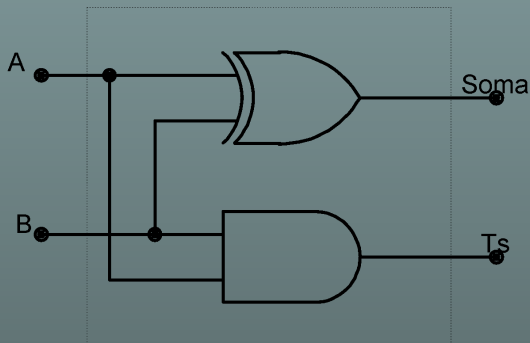
Circuitos aritméticos

- Dentro do conjunto de circuitos combinacionais aplicados para finalidade específica nos sistemas digitais, destacam-se os circuitos aritméticos;
- São utilizados, principalmente, para construir a ULA (Unidade lógica aritmética) dos microprocessadores e, ainda, encontramos disponíveis em circuitos integrados comerciais.

ULA de 1 Bit

- Meio somador

| A | B | S | Ts |
|---|---|---|----|
| 0 | 0 | 0 | 0 |
| 0 | 1 | 1 | 0 |
| 1 | 0 | 1 | 0 |
| 1 | 1 | 0 | 1 |



- Somador completo

| A | B | Ci | S | Co |
|---|---|----|---|----|
| 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 1 | 1 | 0 |
| 0 | 1 | 0 | 1 | 0 |
| 0 | 1 | 1 | 0 | 1 |
| 1 | 0 | 0 | 1 | 0 |
| 1 | 0 | 1 | 0 | 1 |
| 1 | 1 | 0 | 0 | 1 |
| 1 | 1 | 1 | 1 | 1 |

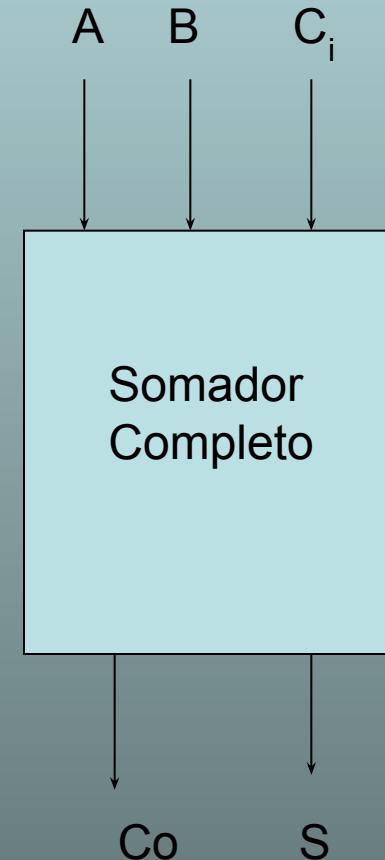
Circuitos Aritméticos Meio Somador

| A | B | S | T _s |
|---|---|---|----------------|
| 0 | 0 | 0 | 0 |
| 0 | 1 | 1 | 0 |
| 1 | 0 | 1 | 0 |
| 1 | 1 | 0 | 1 |



Circuitos Aritméticos – Somador completo

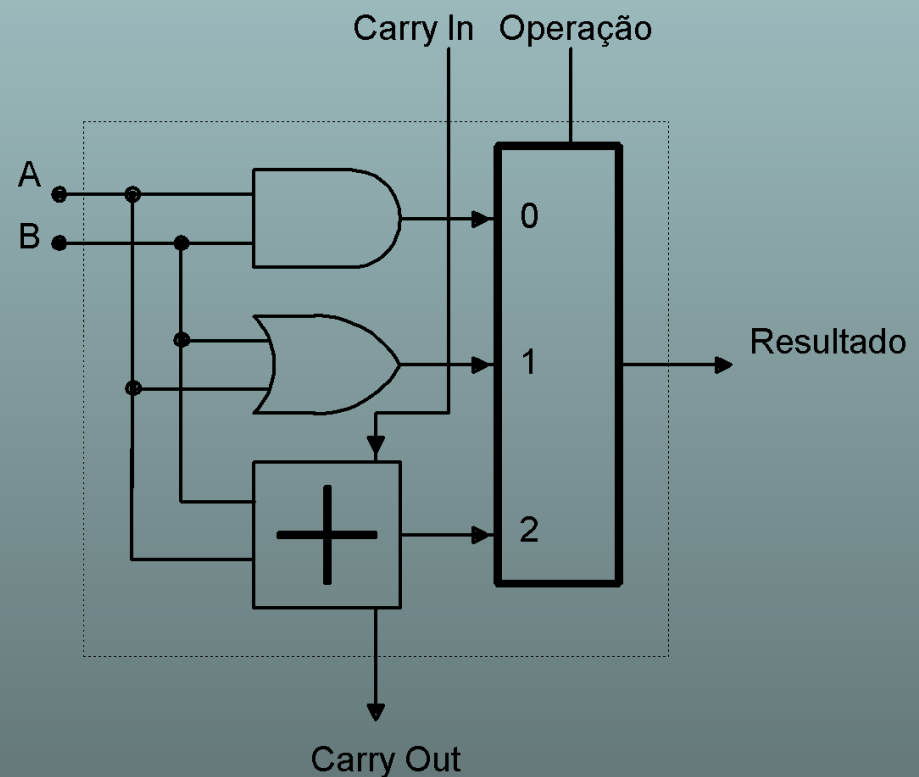
| A | B | C _i | S | C _o |
|---|---|----------------|---|----------------|
| 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 1 | 1 | 0 |
| 0 | 1 | 0 | 1 | 0 |
| 0 | 1 | 1 | 0 | 1 |
| 1 | 0 | 0 | 1 | 0 |
| 1 | 0 | 1 | 0 | 1 |
| 1 | 1 | 0 | 0 | 1 |
| 1 | 1 | 1 | 1 | 1 |



ULA de 1 Bit

- Uma ULA de 1 bit completa pode ser montada como:

- A ULA apresentada é a base das ULAS da maioria dos processadores.



ULA de 32 Bits

- De posse de uma ULA de 1 Bit, é possível obter uma ULA de 32 Bits através da conexão adequada de várias delas.
- A subtração também é realizada com a ajuda do circuito somador através da utilização do complemento a dois.
- Os dados aqui mostrados representam a base da ULA de qualquer computador, mas normalmente outras funções são acrescentadas para realizar diferentes instruções.