

## Codificação/Decodificação com XOR

Mário Leite

...

Na maioria dos cursos de programação, ao abordar os operadores lógicos, normalmente são estudados os três mais conhecidos: **And** (E - conjunção), **OR** (OU - disjunção) e **NOT** (Não - negação). O operador **XOR** (OU exclusivo) é apenas citado com a observação “*este, depois a gente vê*”; mas, na verdade, na maioria das vezes não se vê nenhum exemplo prático com **XOR**; este operador sempre fica relegado a segundo plano. Entretanto, ele é muito importante em sistemas de Segurança de Dados; principalmente em rotinas que utilizam o Método MD5, empregado na codificação de mensagens trocadas entre dois interlocutores. Para uma rápida explicação de como funciona as operações com **XOR** vamos considerar duas expressões lógicas **A** e **B**, mostradas na **tabela 1**, com os dois possíveis resultados lógicos: (**0**-Falso, **1**-Verdadeiro).

A	B	A xor B
0	0	0
1	1	0
0	1	1
1	0	1

Tabela 1 - Tabela-verdade de XOR

Isto é, quando ambas as expressões são verdadeiras (**1**) ou falsas (**0**) o resultado é *falso* (**0**); e quando as duas expressões são diferentes uma da outra o resultado é *verdadeiro* (**1**). E como exemplo PRÁTICO vamos considerar que a mensagem a ser enviada seja **BOTA**: cujas letras têm os seguintes respectivos valores decimais na Tabela ASCII: **66**, **79**, **84** e **65**, com seus valores binários: **01000010**, **01001111**, **01010100** e **01000001**. Então, suponhamos que a chave escolhida (**K**) seja a letra “j” (jota minúsculo), cujo valor decimal é **106** e tendo como representação binária a sequência de *bits*: **01101010**. A **tabela 2** mostra as operações de *codificação/decodificação* da mensagem com o operador **XOR**, sendo **M** esta mensagem.

M ==>	B	O	T	A
	01000010	01001111	01010100	01000001
K (j=106)	01101010	01101010	01101010	01101010
C = M xor K	00101000	00100101	00111110	00101011
C xor K	01000010	01001111	01010100	01000001

Tabela 2 - Codificação/Decodificação de uma mensagem com o operador XOR

Devido à uma propriedade do operador **xor** temos o seguinte:

Se (A xor B = C) Então

C xor A = B

e

C xor B = A

Isto quer dizer que ao receber a mensagem codificada com a operação “Mensagem **xor** Chave”, e sabendo a chave de decodificação, o receptor pode recuperar (*decodificar*) a mensagem, conforme demonstrado na **tabela 2**. Portanto, como pode ser visto, o operador **XOR** é muito importante; não o despreze! O programa “OperacoesBitBit” (em Visualg), mostrado abaixo, apresenta as quatro operações binárias onde a opção da operação **XOR** é destacada com um exemplo tirado da **tabela 2**. A sequências das figuras **1, 2, 3, e 4** ilustram este exemplo de operação.

#### Algoritmo "OperacoesBitBit"

```
//Lê dois números binários (máximo oito bits) e faz as operações com AND, OR, XOR e NOT.  
//Autor: Mario Leite  
//E-mail: marleite@gmail.com  
//-----
```

```
Const TamBin=8 //define o padrão da palavra para 1 byte (8 bits)  
Var VetBin1N, VetBin2N, VetRetN: vetor[1..8] de inteiro  
    VetBin1S, VetBin2S, VetRetS: vetor[1..8] de caractere  
    j, Op, TamBin1, TamBin2, DifTam1, DifTam2: inteiro  
    nada, Aux1, Aux2, Bin, Bin1, Bin2, RetStr, BitAux, BitRet: caractere  
    Valido, Cond: logico
```

#### Inicio

Repita

  LimpaTela

  Escreval("") //salta linha

  {Menu de opções para as operações desejadas}

  Op <- 0

  Escreval("Operações binárias bit a bit")

  Escreval("-----")

  Escreval("Operação AND..... 1")

  Escreval("Operação OR..... 2")

  Escreval("Operação XOR..... 3")

  Escreval("Operação NOT..... 4")

  Escreval("Encera o programa..... 5")

  Escreval("-----")

  Escreval("")

Repita

  Escreva("Digite a sua opção: ")

  Leia(Op)

Ate((Op>=1) e (Op<=5))

Se(Op=5) Entao

  Interrompa //abandona o loop

FimSe

Se(Op<>4) Entao

  Escreval("")

  {Lê o primeiro número binário}

  Repita

    Valido <- Verdadeiro

    Repita

      Escreva("Digite o primeiro número binário[tam min 4-tam máx",TamBin,"]: ")

      Leia(Bin1)

      TamBin1 <- Compr(Bin1)

    Ate((TamBin1>=4) e (TamBin1<=TamBin))

    {Lê o segundo número binário}

    Repita

      Escreva("Digite o segundo número binário [tam min 4-tam máx",TamBin,"]: ")

      Leia(Bin2)

      TamBin2 <- Compr(Bin2)

    Ate((TamBin2>=4) e (TamBin2<=TamBin))

```

{Valida cada bit do primeiro número binário}
Aux1 <- ""
Para j De 1 Ate TamBin1 Faca
    VetBin1S[j] <- Copia(Bin1,j,1) //cria vetor de elementos caracteres
    Aux1 <- Aux1 + VetBin1S[j] //cria string com elementos do primeiro binário
    VetBin1N[j] <- CaracpNum(VetBin1S[j]) //cria vetor de elementos numéricos
    Cond <- (VetBin1S[j]<>"1") e (VetBin1S[j]<>"0")
    Se(Cond) Entao
        Valido <- Falso
        Interrompa //algum bit de Bin1 é inválido
    FimSe
FimPara
{Valida cada bit do segundo número binário}
Aux2 <- ""
Para j De 1 Ate TamBin2 Faca
    VetBin2S[j] <- Copia(Bin2,j,1) //cria vetor de elementos caracteres
    Aux2 <- Aux2 + VetBin2S[j] //cria string com elementos do segundo binário
    VetBin2N[j] <- CaracpNum(VetBin2S[j]) //cria vetor de elementos numéricos
    Cond <- (VetBin2S[j]<>"1") e (VetBin2S[j]<>"0")
    Se(Cond) Entao
        Valido <- Falso
        Interrompa //algum bit de Bin2 é inválido
    FimSe
FimPara
Se(Nao(Valido)) Entao
    Escreval("")
    Escreval("Dígito inválido em algum dos números")
FimSe
Ate((Valido) e (TamBin1>=4) e (TamBin1<=8) e (TamBin2>=4) e (TamBin2<=8))
{Verifica os tamanhos dos binários ver se precisa de preencher com 0 à esquerda}
DifTam1 <- TamBin - Abs(TamBin1)
DifTam2 <- TamBin - Abs(TamBin2)
Se(DifTam1 < TamBin) Entao
    Para j De 1 Ate DifTam1 Faca
        Aux1 <- "0" + Aux1
    FimPara
    {Recria o primeiro vetor para o tamanho TamBin}
    Para j De 1 Ate TamBin Faca
        VetBin1S[j] <- Copia(Aux1,j,1)
    FimPara
FimSe
Se(DifTam2 < TamBin) Entao
    Para j De 1 Ate DifTam2 Faca
        Aux2 <- "0" + Aux2
    FimPara
    {Recria o segundo vetor para o tamanho TamBin}
    Para j De 1 Ate TamBin Faca
        VetBin2S[j] <- Copia(Aux2,j,1)
    FimPara
FimSe
{Realiza a operação de acordo com a opção}
Escolha Op
Caso 1
    {Faz a operação Bin1 AND Bin2}
    RetStr <- ""
    Para j De 1 Ate TamBin Faca
        Cond <- ((VetBin1S[j]="1") e (VetBin2S[j]="1"))
        Se(Cond) Entao
            VetRetS[j] <- "1"
        Senao
            VetRetS[j] <- "0"
    FimSe

```

```

    RetStr <- RetStr + VetRetS[j]
FimPara
Escreval("")
Escreval("Operação: (",Bin1, " AND ", Bin2,") = ",RetStr)
Caso 2
    {Faz a operação Bin1 OR Bin2}
    RetStr <- ""
    Para j De 1 Ate TamBin Faca
        Cond <- ((VetBin1S[j]="1") ou (VetBin2S[j]="1"))
        Se(Cond) Entao
            VetRetS[j] <- "1"
        Senao
            VetRetS[j] <- "0"
        FimSe
        RetStr <- RetStr + VetRetS[j]
    FimPara
    Escreval("")
    Escreval("Operação: (",Bin1, " OR ", Bin2,") = ",RetStr)
Caso 3
    {Faz a operação Bin1 XOR Bin2}
    Para j De TamBin Ate 1 Passo -1 Faca
        Cond <- (VetBin1S[j]<>VetBin2S[j])
        Se(Cond) Entao
            VetRetS[j] <- "1"
        Senao
            VetRetS[j] <- "0"
        FimSe
        RetStr <- RetStr + VetRetS[j]
    FimPara
    RetStr <- ""
    Para j De 1 Ate TamBin Faca
        RetStr <- RetStr + VetRetS[j]
    FimPara
    Escreval("")
    Escreval("Operação: (",Bin1, " XOR ", Bin2,") = ",RetStr)
FimEscolha
Senao
    {Faz a operação NOT(Bin)}
    Escreva("Digite o número binário [tam min 4 - tam máx",TamBin,"]: ")
    Leia(Bin)
    RetStr <- ""
    Para j De 1 Ate TamBin Faca
        BitAux <- Copia(Bin,j,1)
        Se(BitAux="1") Entao
            BitAux <- "0"
        Senao
            BitAux <- "1"
        FimSe
        RetStr <- RetStr + BitAux
    FimPara
    Escreval("Operação: NOT(",Bin,") = ", RetStr)
FimSe
Escreval("")
Escreval("")
Escreval("")
Escreva("Pressione a tecla <Enter> para continuar...")
Leia(nada)
LimpaTela
Ate (op=5)
FimAlgoritmo

```

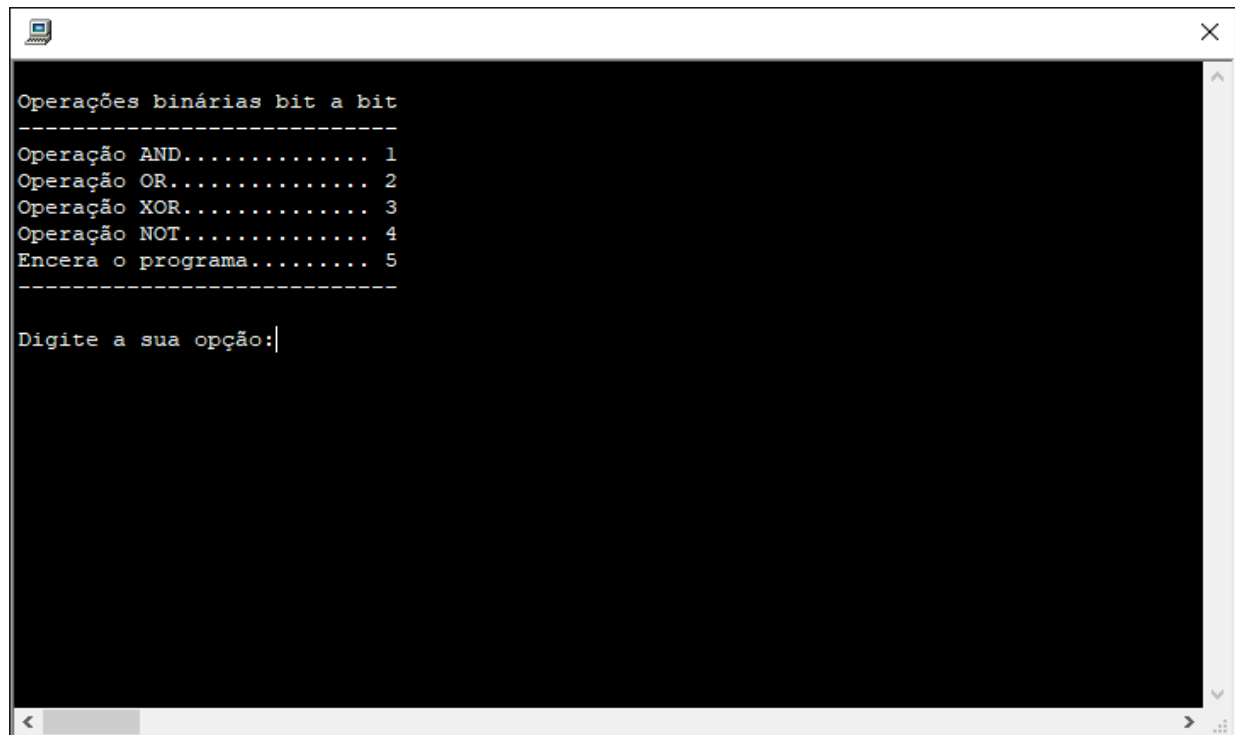


Figura 1 - Ao carregar o programa: aparece o menu de opções

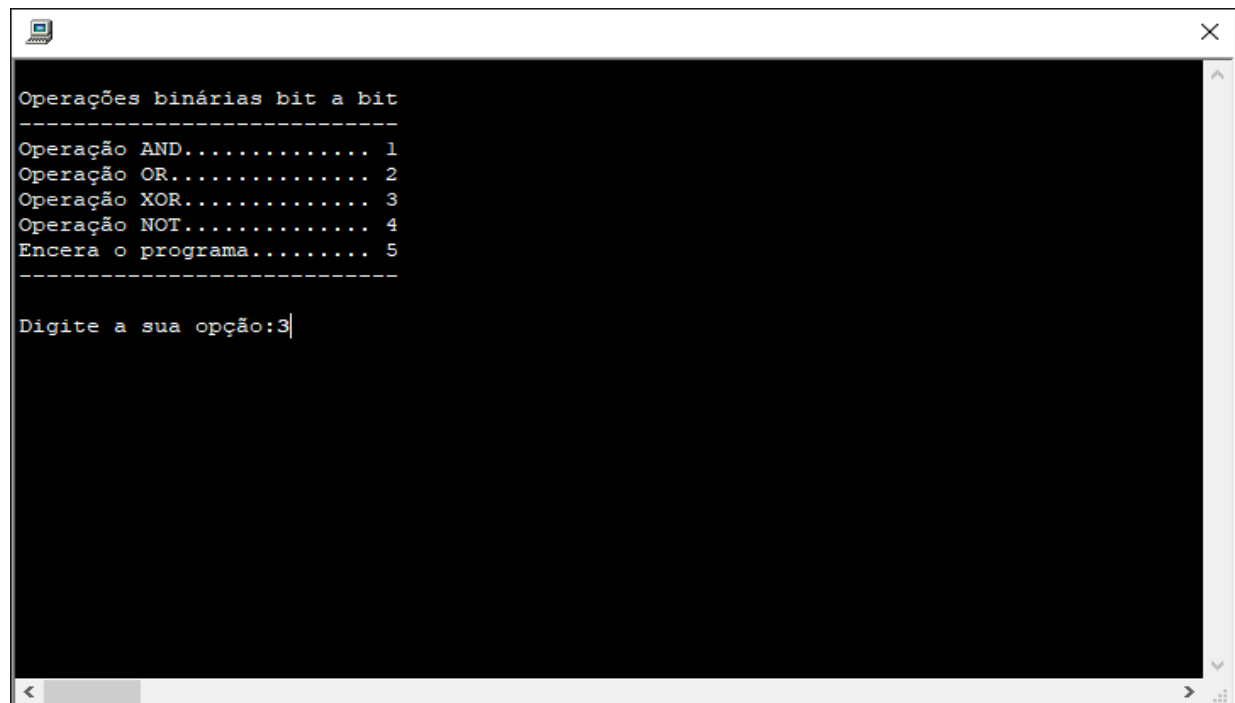
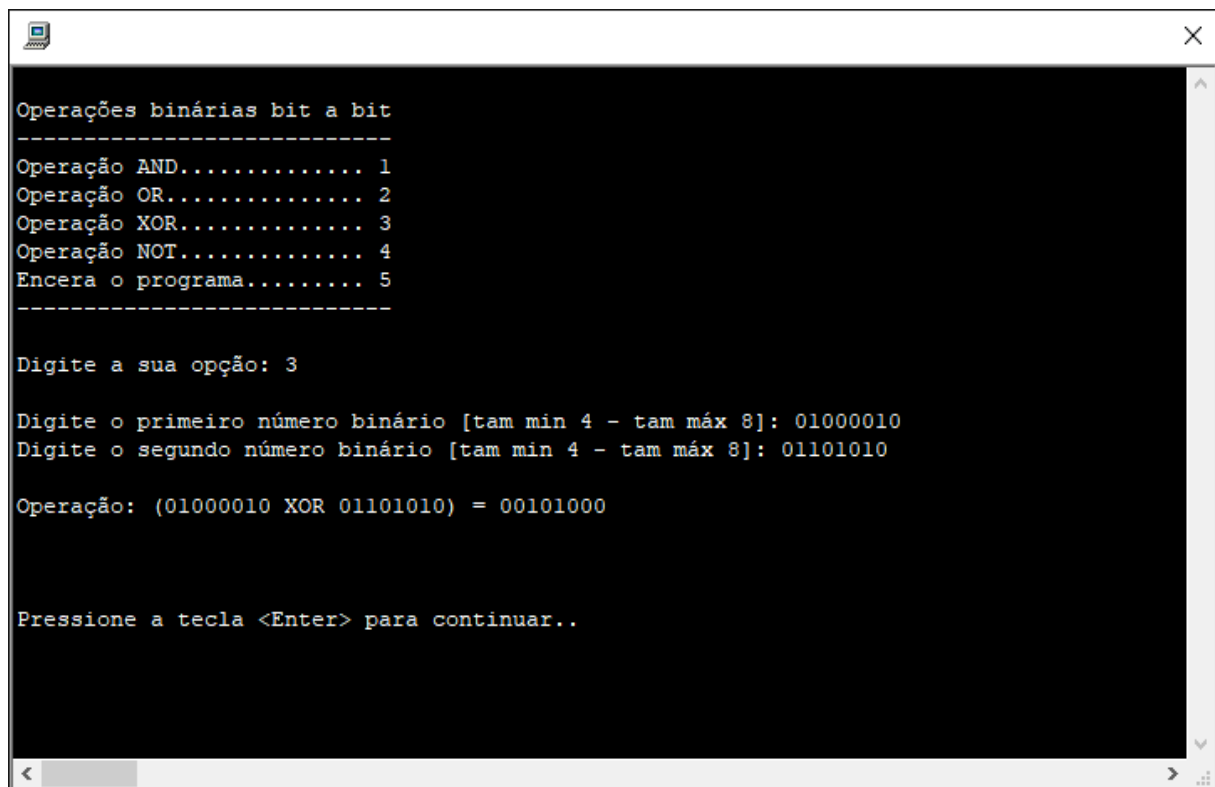


Figura 2 - Escolhendo a opção 3 (operação XOR)



```
Operações binárias bit a bit
-----
Operação AND..... 1
Operação OR..... 2
Operação XOR..... 3
Operação NOT..... 4
Encera o programa..... 5
-----

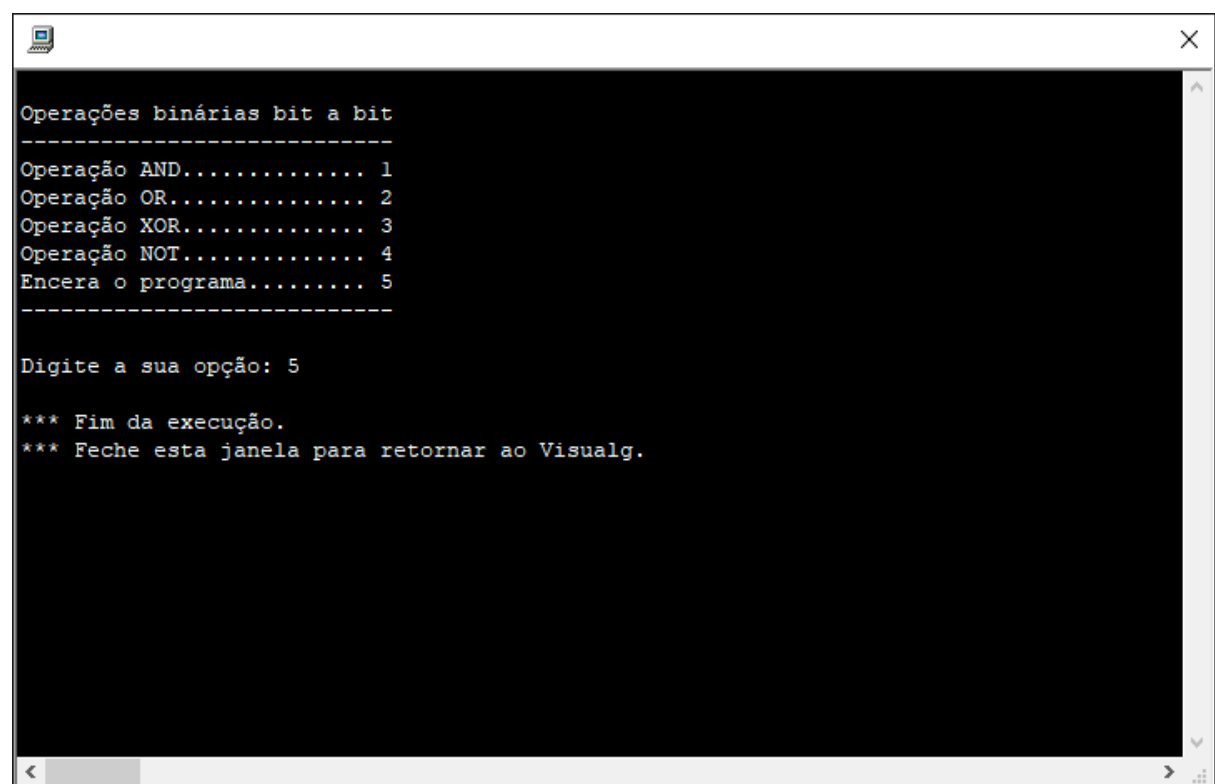
Digite a sua opção: 3

Digite o primeiro número binário [tam min 4 - tam máx 8]: 01000010
Digite o segundo número binário [tam min 4 - tam máx 8]: 01101010

Operação: (01000010 XOR 01101010) = 00101000

Pressione a tecla <Enter> para continuar..
```

Figura 3 - Entrando com os dados e obtendo o resultado



```
Operações binárias bit a bit
-----
Operação AND..... 1
Operação OR..... 2
Operação XOR..... 3
Operação NOT..... 4
Encera o programa..... 5
-----

Digite a sua opção: 5

*** Fim da execução.
*** Feche esta janela para retornar ao Visualg.
```

Figura 4 - Escolhendo a opção 5 para encerrar o programa