

Instruções lógicas

Instruções lógicas

- São 5 instruções lógicas: **or**, **and**, **not**, **xor** e **test**

Example				
Before	Instruction	Bitwise Operation	After	
AX: E2 75 CX: A9 D7	and ax, cx	1110 0010 0111 0101 <u>1010 1001 1101 0111</u> 1010 0000 0101 0101	AX	<div>A055</div> SF 1 ZF 0
DX: E2 75 value: A9 D7	or dx, value	1110 0010 0111 0101 <u>1010 1001 1101 0111</u> 1110 1011 1111 0111	DX	<div>EBF7</div> SF 1 ZF 0
BX: E2 75	xor bx, 0a9d7h	1110 0010 0111 0101 <u>1010 1001 1101 0111</u> 0100 1011 1010 0010	BX	<div>4BA2</div> SF 0 ZF 0
AX: E2 75	not ax	<u>1110 0010 0111 0101</u> 0001 1101 1000 1010	AX	<div>1D8A</div>

A instrução **and**

- A instrução **and** é útil em três situações:
 - suportar expressões lógicas e operação bit a bit de linguagens de alto nível
 - limpar um ou mais bits
 - isolar um ou mais bits

A instrução **and**

- Limpando bits através de uma mascara:

```
AL = 11010110 ← operand to be manipulated
BL = 11111100 ← mask byte
and  AL,BL = 11010100
```

- Utilizando o bit mais significativo para ser o bit de paridade:

```
and    AL, 7FH
```

A instrução **and**

- Isolando bits para teste:
 - verifica se um número é par ou ímpar (ZF é configurado se o número for par):

```
        and     AL,1        ; mask = 00000001B
        jz      even_number
odd number:
        . . .
        <code for processing odd number>
        . . .
even_number:
        . . .
        <code for processing even number>
        . . .
```

A instrução **or**

- A instrução **or** é útil em duas situações:
 - suportar expressões lógicas e operação bit a bit de linguagens de alto nível
 - configurar um ou mais bits:

AL = 11010110B ← operand to be manipulated

BL = 00000011B ← mask byte

or AL, BL = 11010111B

A instrução **or**

- Codificação do bit de paridade par

`or AL, 80H`

- Se a quantidade de bits 1 nos sete bits menos significativos for ímpar configura o bit mais significativo para 1.

A instrução **or**

- Recorta e cola bits:

```
and    AL, 55H    ; cut odd bits
and    BL, 0AAH   ; cut even bits
or     AL, BL     ; paste them together
```


A instrução **xor**

- A instrução **xor** é útil em três situações:
 - suportar expressões lógicas de linguagens de alto nível
 - alternar um ou mais bits
 - inicializar registradores

A instrução **xor**

- Alternando o bit de paridade:

`xor AL, 80H`



	01000001B	← even-parity encoded ASCII character A
xor	<u>10000000B</u>	← mask byte
	11000001B	← odd-parity encoded ASCII character A

A instrução **xor**

- Criptografando dados:

```
; read a data byte into AL
xor    AL,0A6H
; write the data byte back from AL
```



01000010B	← ASCII character B
00100110B	← encryption key (mask)
<u>01000010B</u>	
01100100B	← ASCII character d

A instrução **xor**

- Pode ser utilizado também para inicializar registradores:

`mov EAX, 0`



Equivale a

`xor EAX, EAX`

A instrução **not**

- É utilizada para complementar bits (complemento de 1):

```
not    AL
inc    AL
```

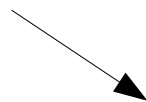
- suporta também operações lógicas e operação bit a bit de linguagens de alto nível

A instrução **test**

- Realiza um **and** sem alterar o registrador de destino.

```
and    AL, 1    ;Altera AL
```

;não altera **AL**



```
test    AL, 1    ; mask = 00000001B
jz      even_number
odd_number:
        . . .
even_number:
        . . .
```

Instruções lógicas de deslocamento

- Realizam deslocamento de bits. CF, ZF e OF são afetados

`shl` (SHift Left)

`shr` (SHift Right)

C statement	Assembly language instruction
<code>mask = mask >> 2</code> (right-shift mask by two bit positions)	<code>shr SI, 2</code>
<code>mask = mask << 4</code> (left-shift mask by four bit positions)	<code>shl SI, 4</code>


Instruções lógicas de deslocamento

- Utilizada para criptografia:

AH = AL = 01000001B



```
; AL contains the byte to be encrypted
mov     AH,AL
shl     AL,4      ; move lower nibble to upper
shr     AH,4      ; move upper nibble to lower
or      AL,AH     ; paste them together
; AL has the encrypted byte
```



AL = 00010000B
AH = 00000100B

or AL,AH = 00010100B

Instruções lógicas de deslocamento

- Pode ser utilizado também para multiplicar e dividir números por 2.

Binary number	Decimal value
00011100	28
00111000	56
01110000	112
11100000	224
10101000	168
01010100	84
00101010	42
00010101	21

Exercícios

- Defina os valores dos registradores após a execução das instruções abaixo:

<i>Before</i>	<i>Instruction</i>	<i>After</i>
(a) BX: FA 75 CX: 31 02	and bx, cx	BX, SF, ZF
(b) BX FA 75 CX 31 02	or bx, cx	BX, SF, ZF
(c) BX FA 75 CX 31 02	xor bx, cx	BX, SF, ZF
(d) BX FA 75	not bx	BX
(e) AX FA 75	and ax, 000fh	AX, SF, ZF
(f) AX FA 75	or ax, 0fff0h	AX, SF, ZF
(g) AX FA 75	xor ax, 0ffffh	AX, SF, ZF
(h) AX FA 75	test ax, 0004h	AX, SF, ZF