

## 6. Instruções lógicas, de deslocamento e de rotação

- São instruções que permitem **mudar o padrão de bits** num byte (8 bits) ou numa palavra (16 bits).
- Linguagens de alto nível (exceto C) **não permitem** manipular diretamente bits.
- Instruções lógicas **AND, OR, XOR e NOT** são usadas para:
  - **resetar** (*reset*) ou **limpar** (*clear*) um bit: 1 -> 0
  - **setar** (*set*) um bit: 0 -> 1
  - examinar bits
  - realizar **máscaras** para manipular bits

### Operadores lógicos:

a	b	a AND b	a OR b	a XOR b
0	0	0	0	0
0	1	0	1	1
1	0	0	1	1
1	1	1	1	0

a	NOT a
0	1
1	0

Obs: em bytes ou palavras, os operadores lógicos são **aplicados bit a bit**.

- Instruções de deslocamento (*shift*):
  - deslocar para a esquerda 1 casa binária => multiplicar por dois
  - deslocar para a direita 1 casa binária => dividir por dois
  - os bits **deslocados para fora** são perdidos
- Instruções de rotação (*rotate*):
  - deslocar de forma circular (em anel) para a esquerda ou para a direita
  - **nenhum** bit é perdido

## 6.1 Instruções lógicas

**AND destino,fonte**  
**OR destino,fonte**  
**XOR destino,fonte**

Usadas para aplicar os operadores lógicos correspondentes bit a bit entre:

- registrador e registrador
- registrador e uma posição de memória
- o operando fonte pode ser também uma constante

Combinações legais de operandos:

Operando fonte	Operando destino	
	Registrador de dados	Posição de memória
Reg. de dados	sim	sim
Posição de memória	sim	<b>não</b>
Constante	sim	sim

- **Flags afetados:**

SF, ZF, PF refletem o resultado (armazenado no operando destino)

AF não é afetado

CF e OF ficam em **zero**, ou seja, são **resetados**

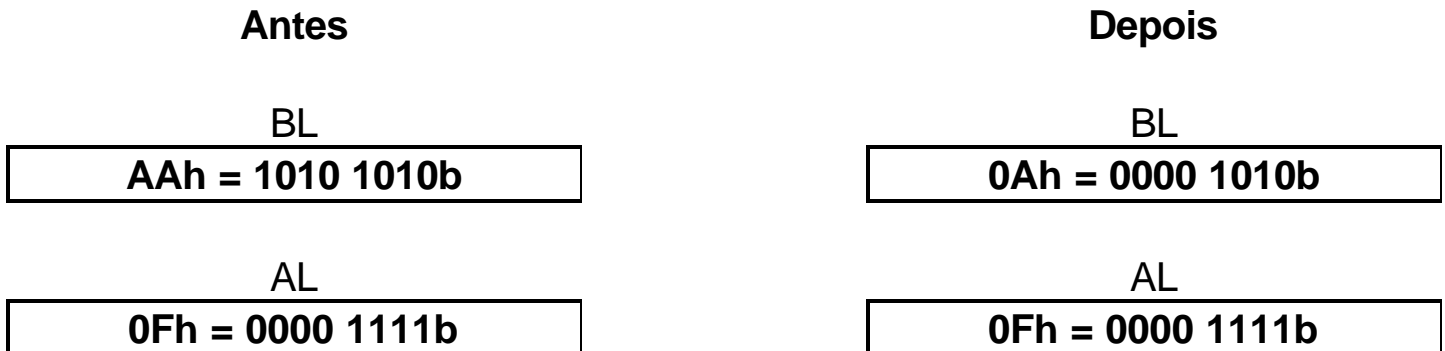
Exemplos de instruções válidas:

XOR AX,BX ;operador XOR aplicado aos conteúdos de AX e BX,  
;resultado em AX

AND CH,01h ;operador AND aplicado ao conteúdo de CH, tendo  
;como fonte o valor imediato 01h = 0000 0001b

OR WORD1,BX;operador OR entre conteúdos da posição de memória  
;WORD1 e de BX, resultado armazenado em WORD1

Graficamente: suponha a instrução **AND BL,AL**



**Obs:** Propriedades dos operadores lógicos aplicados bit a bit:

bit(x) AND 0 = 0	bit(x) AND 1 = bit(x)
bit(x) OR 0 = bit(x)	bit(x) OR 1 = 1
bit(x) XOR 0 = bit(x)	bit(x) XOR 1 = complemento do bit(x)

**Criação de máscaras:** padrão de "0" e "1" para manipular bits por meio de operações lógicas.

- **AND** pode ser utilizado para **zerar** (*clear* ou *reset*) bits específicos: basta ter um **0** na posição que se deseja este efeito.
- **OR** pode ser utilizado para **setar** (*set*) bits específicos: deve-se ter um **1** na posição em que se deseja este efeito.
- **XOR** pode ser utilizado para **complementar** (inverter) bits específicos: deve-se ter um **1** na posição em que se deseja este efeito.

## Exemplos de máscaras:

1) Setar os bits MSB e LSB do registrador AX, dado AX = 7444h:

**OR AX,8001h**

AX (antes)	->	0111 0100 0100 0100b	->	7444h
8001h	->	1000 0000 0000 0001b		
		OR		
AX (depois)	->	<u>1111 0100 0100 0101b</u>	->	F445h

2) Convertendo o código ASCII de um dígito numérico em seu valor binário:

**AND AL,0Fh** (em substituição a: SUB AL,30h)

AL (antes)	->	0011 0111b	->	37h = "7" = 55d
0Fh	->	0000 1111b		
		AND		
AL (depois)	->	<u>0000 0111b</u>	->	07h = 7d (valor <b>sete</b> )

3) Convertendo letra minúscula em maiúscula, supondo o caracter em AL:

**AND AL,0DFh**

AL (antes)	->	0110 0001b	->	61 h = "a"
DFh	->	1101 1111b		
		AND		
AL (depois)	->	<u>0100 0001b</u>	->	41h = "A"

Obs: para esta conversão, tem-se apenas que **zerar** (resetar) o **bit 5** de AL.

## Mais exemplos de aplicação de operações lógicas:

1) Limpando (zerando) um registrador:

**XOR AX, AX**

AX (antes)	->	0111 0100 0100 0100b	->	7444h
		0111 0100 0100 0100b		
		XOR		
AX (depois)	->	<b>0000 0000 0000 0000b</b>	->	0000h = 0

Obs: esta forma é mais rápida de executar do que as outras opções

**MOV AX,0000h    e    SUB AX,AX**

2) Testando se o conteúdo de algum registrador é zero:

**OR CX,CX**

CX (antes)	->	0111 0100 0100 0100b	->	7444h
		0111 0100 0100 0100b		
		OR		
CX (depois)	->	0111 0100 0100 0100b	->	7444h (não é 0)

Obs:

- esta operação deixa o registrador CX inalterado
- modifica o FLAG **ZF somente** quando o conteúdo de CX é realmente zero
- esta forma é mais rápida de executar do que **CMP CX,0000h**.

## NOT destino

Usada para aplicar o operador lógico NOT em todos os bits de:

- um registrador
- uma posição de memória
- o resultado é a **complementação** (inversão) de todos os bits
  
- **Flags afetados:** nenhum

Exemplos de instruções válidas:

NOT AX ;inverte todos os bits de AX

NOT AL ;inverte todos os bits de AL

NOT BYTE1 ;inverte todos os bits do conteúdo da posição de  
;memória definida pelo nome BYTE1

Graficamente: suponha a instrução **NOT WORD1**

**Antes**

BYTE1

**81h = 1000 0001b**

**Depois**

BYTE1

**7Eh = 0111 1110b**

## TEST destino,fonte

Usada para aplicar o operador lógico AND entre:

- registrador e registrador
- registrador e uma posição de memória
- o operando fonte pode ser também uma constante

**sem afetar** o operando destino (**não armazena** o resultado do AND).

Combinações legais de operandos:

Operando fonte	Operando destino	
	Registrador de dados	Posição de memória
Reg. de dados	sim	sim
Posição de memória	sim	<b>não</b>
Constante	sim	sim

- **Flags afetados:**

SF, ZF, PF refletem o resultado (armazenado no operando destino)

AF não é afetado

CF e OF ficam em **zero**

Exemplos de instruções válidas:

TEST AX,BX                   ;operação AND entre AX e BX, não há resultado, mas  
;apenas alteração dos FLAGS   ZF, SF e PF

TEST AL,01h                 ;operação AND entre AL e o valor imediato 01h

Graficamente: suponha a instrução **TEST AX,0001h**

Antes	Depois
AX	AX
4444h = 0100 0100 0100 0100b	0100 0100 0100 0100b
ZF	ZF
0	1

Neste exemplo, a máscara **0001h** serve para testar se o conteúdo de AX é **PAR** (todo número binário PAR possui um **zero** no **LSB**)

- O número 4444h é PAR pois o seu LSB vale zero
- 4444h **AND** 0001h produz como resultado 0000h que faz **ZF = 1**
- o resultado não é armazenado em AX, somente ZF é modificado por TEST

**Exemplo:** escreva um trecho de programa que salte para o rótulo PONTO2 se o conteúdo de CL for negativo:

```
....
TEST CL,80h      ;80h é a máscara 1000 0000b
JNZ PONTO2
```

```
....
(o programa prossegue, pois o número é positivo)
```

```
....
```

PONTO2: ....

```
(o programa opera aqui com o número negativo)
```

```
....
```



**Exercícios sugeridos:**

1) Dê a operação lógica e a máscara correspondente para cada item abaixo, tal que:

- a) Limpe (*clear*) os bits pares de AX, deixando os restantes sem alteração;
- b) Sete (*set*) o MSB de BL, permanecendo os restantes como estão;
- c) Complemente o MSB de DX, deixando os restantes intocados;
- d) Remova os quatro bits mais significativos de AL, permanecendo os restantes inalterados;
- e) Faça o complemento de 2 de BX sem utilizar a instrução NEG (talvez seja necessário um trecho de programa ao invés de uma única operação lógica).

2) Use a instrução **TEST** em cada item abaixo para:

- a) Fazer ZF = 1 caso o conteúdo de AX seja zero;
- b) Zerar ZF caso o conteúdo de BL seja IMPAR;
- c) Setar SF caso o conteúdo de DX seja um número negativo;
- d) Setar ZF caso o conteúdo de DX seja zero ou um número positivo;
- e) Setar PF caso BL contenha um número PAR de bits 1.



## Mecânica de deslocamento

(página colada, referir à apostila, disponível no Xerox)

**Exemplos:**

1) Multiplique o conteúdo de AX por 8, supondo AX contendo 0005h = 5d:

AX (original)	0000 0000 0000 <b>0101</b>	= 5d
após o 1o. deslocamento	0000 0000 0000 <b>1010</b>	= 10d
após o 2o. deslocamento	0000 0000 0001 <b>0100</b>	= 20d
após o 3o. deslocamento	0000 0000 0010 <b>1000</b>	= 40d

trecho de programa:

```

...
MOV    CL,3
SHL    AX,CL    ;poderia ser SAL AX,CL
...

```

2) Divida o conteúdo de DH por 4, supondo DH contendo 12h = 18d:

DH (original)	0001 <b>0010</b>	= 18d
após o 1o. deslocamento	0000 <b>1001</b>	= 9d
após o 2o. deslocamento	0000 0 <b>100</b>	= 4d      CF = 1

Obs: pelo fato do número 9 ser impar, sua divisão por dois foi arredondada para o inteiro imediatamente inferior.

trecho de programa:

```

...
MOV    CL, 02
SAR    DH,CL

```

...

### 6.3 Instruções de rotação:

**Rxx destino, 1**  
**Rxx destino, CL**

Usada para rodar (deslocar em anel) para a **esquerda** ou para a **direita**:

- 1 bit
- tantos quantos CL indicar

- um registrador
- uma posição de memória

<b>Rxx</b>	<b>Significado</b>
ROL	<i>Rotate Left</i> - rodar para a esquerda
ROR	<i>Rotate Right</i> - rodar para a direita
RCL	<i>Rotate Carry Left</i> - rodar para a esquerda através do flag CF
RCR	<i>Rotate Carry Right</i> - rodar para a direita através do flag CF

- **Flags afetados:**

SF, ZF, PF refletem o resultado da última rotação

AF não é afetado

CF contem o último bit deslocado para fora

OF = 1 se ocorrer troca de sinal após a última rotação

Exemplos de instruções válidas:

ROL AX,1                   ;desloca os bits de AX para a esquerda 1 casa binária,  
                               ;sendo o MSB é **reinserido** na posição LSB

ROR BL,CL                 ;desloca os bits de BL para a direita tantas casas  
                               ;binárias quantas CL indicar, os bits menos  
                               ;significativos são reinseridos um-a-um no MSB

RCR DH,1                 ;desloca os bits de DH para a direita 1 casa binária,  
                               ;sendo que o MSB recebe CF e o LSB é salvo em CF

## **Mecânica de rotação**

(2 páginas coladas, referir à apostila, disponível no Xerox)

**Exemplos:**

1) Conte o número de bits "1's" presentes em AX, sem destruí-lo:

Utilizando:

- CX como contador de bits
- BL como contador de "1's"

```

...
    XOR BL,BL           ;inicializa-se BL com zero
    MOV CX,16          ;inicializa-se contador de bits (AX tem 16 bits)
TOPO:  ROL AX, 1       ;roda-se AX 1 casa a esquerda e
                    ;CF contem uma cópia do bit deslocado
    JNC PT1            ;se CF = 1
    INC BL              ;conta-se 1 em BL
PT1:   LOOP TOPO       ;senão, processegue-se no laço
...
                    ;até que CX = 0

```

2) Invertendo o padrão de bits de AL:

Se AL (antes) = 1000 1100b, AL (depois) = 0011 0001b

```

...
    XOR BL,BL           ;inicializa-se BL com zero
    MOV CX,8            ;inicializa-se contador de bits (AL tem 8 bits)
TOPO:  SHL AL,1         ;desloca-se AL 1 casa para a esquerda
                    ;CF contem o bit deslocado para fora
    RCR BL,1            ;roda-se BL para a direita através de CF
    LOOP TOPO           ;enquanto CX não for zero, repete TOPO
    MOV AL,BL           ;quando CX = 0, AL recebe o padrão
...

```



## 6.4 Entrada e saída de números binários e hexadecimais:

### . Entrada de números binários:

- *string* de caracteres "0's" e "1's" fornecidos pelo teclado;
- CR é o marcador de fim de *string*;
- BX é assumido como registrador de armazenamento;
- máximo de 16 bits de entrada.

### Algoritmo básico em linguagem de alto nível:

Limpa BX

Entra um caracter "0" ou "1"

WHILE caracter diferente de CR DO

    Converte caracter para valor binário

    Desloca BX 1 casa para a esquerda

    Insere o valor binário lido no LSB de BX

    Entra novo caracter

END\_WHILE

### Trecho de programa implementado em Linguagem Montadora:

```

...
    MOV CX,16           ;inicializa contador de dígitos
    MOV AH,1h          ;função DOS para entrada pelo teclado
    XOR BX,BX          ;zera BX -> terá o resultado
    INT 21h            ;entra, caracter está no AL
;while
    TOPO:  CMP AL,0Dh    ;é CR?
           JE  FIM      ;se sim, termina o WHILE
           AND AL,0Fh   ;se não, elimina 30h do caracter
                           ;(poderia ser SUB AL,30h)
           SHL BX,1     ;abre espaço para o novo dígito
           OR  BL,AL    ;insere o dígito no LSB de BL
           INT 21h     ;entra novo caracter
           LOOP TOPO   ;controla o máximo de 16 dígitos
;end_while
    FIM:  ...

```

- **Saída de números binários:**

- BX é assumido como registrador de armazenamento;
- total de 16 bits de saída;
- *string* de caracteres "0's" e "1's" é exibido no monitor de vídeo.

### Algoritmo básico em linguagem de alto nível:

```

FOR 16 vezes DO
    rotação de BX à esquerda 1 casa binária (MSB vai para o CF)
    IF CF = 1
        THEN exibir no monitor caracter "1"
        ELSE exibir no monitor caracter "0"
    END_IF
END_FOR

```

### Trecho de programa implementado em Linguagem Montadora:

```

...
MOV CX,16           ;inicializa contador de bits
MOV AH,02h         ;prepara para exibir no monitor
;for 16 vezes do
    PT1: ROL BX,1    ;desloca BX 1 casa à esquerda
    ;if CF = 1
        JNC PT2     ;salta se CF = 0
    ;then
        MOV DL, 31h ;como CF = 1
        INT 21h     ;exibe na tela "1" = 31h
    ;else
        PT2: MOV DL, 30h ;como CF = 0
        INT 21h     ;exibe na tela "0" = 30h
    ;end_if
        LOOP PT1    ;repete 16 vezes
;end_for

```

...

• **Entrada de números hexadecimais:**

- BX é assumido como registrador de armazenamento;
- *string* de caracteres "0" a "9" ou de "A" a "F", digitado no teclado;
- máximo de 16 bits de entrada ou máximo de 4 dígitos hexa.

**Algoritmo básico em linguagem de alto nível:**

Inicializa BX

Entra um caracter hexa

WHILE caracter diferente de CR DO

    Converte caracter para binário

    Desloca BX 4 casas para a esquerda

    Insere valor binário nos 4 bits inferiores de BX

    Entra novo caracter

END\_WHILE

**Trecho de programa implementado em Linguagem Montadora:**

```

...
    XOR BX,BX           ;inicializa BX com zero
    MOV CL,4           ;inicializa contador com 4
    MOV AH,1h         ;prepara entrada pelo teclado
    INT 21h           ;entra o primeiro caracter
;while
    TOPO:  CMP AL,0Dh   ;é o CR ?
           JE  FIM
           CMP AL, 39h ;caracter número ou letra?
           JG  LETRA   ;caracter já está na faixa ASCII
           AND AL,OFh  ;número: retira 30h do ASCII
           JMP DESLOC
    LETRA:  SUB AL,37h  ;converte letra para binário
    DESLOC: SHL BX,CL  ;desloca BX 4 casas à esquerda
           OR  BL,AL   ;insere valor nos bits 0 a 3 de BX
           INT 21h    ;entra novo caracter
           JMP TOPO   ;faz o laço até que haja CR
;end_while
    FIM:    ...

```

- **Saída de números hexadecimais:**

- BX é assumido como registrador de armazenamento;
- total de 16 bits de saída;
- *string* de caracteres HEXA é exibido no monitor de vídeo.

**Algoritmo básico em linguagem de alto nível:**

```
FOR 4 vezes DO
  Mover BH para DL
  Deslocar DL 4 casas para a direita
  IF DL < 10
    THEN converte para caracter na faixa 0 a 9
    ELSE converte para caracter na faixa A a F
  END_IF
  Exibição do caracter no monitor de vídeo
  Rodar BX 4 casas à esquerda
END_FOR
```

## Trecho de programa implementado em Linguagem Montadora:

```

...                               ;BX já contem número binário
MOV CH,4                          ;CH contador de caracteres hexa
MOV CL,4                          ;CL contador de deslocamentos
MOV AH,2h                         ;prepara exibição no monitor
;for 4 vezes do
    TOPO: MOV DL,BH                ;captura em DL os oito bits mais
                                ;significativos de BX
                                SHR DL,CL
                                ;resta agora em DL somente os 4
                                ;bits mais significativos de BX
;if DL , 10
    CMP DL, 0Ah                  ;testa se é letra ou número
    JAE LETRA
;then
    ADD DL,30h                   ;é número: soma-se 30h
    JMP PT1
;else
    LETRA: ADD DL,37h            ;ao valor soma-se 37h -> ASCII
;end_if
    PT1: INT 21h                 ;exibe
    ROL BX,CL                    ;roda BX 4 casas para a direita
    DEC CH                       ;faz o FOR 4 vezes
    JNZ TOPO
;end_for
...                               ;programa continua

```

## Exercícios sugeridos:

1) Suponha as condições iniciais  $AL = 11001011b$  e  $CF = 1$ . Dê o novo conteúdo de  $AL$  após cada uma das seguintes instruções, sempre com base nas condições iniciais acima:

- a) `SHL AL,1`
- b) `SHR AL,1`
- c) `ROL AL,CL` ;CL contendo 2
- d) `ROR AL,CL` ;CL contendo 3
- e) `SAR AL,CL` ;CL contendo 2
- f) `RCL AL,1`
- g) `RCR AL,CL` ;CL contendo 3

2) Escreva um programa que peça ao usuário para entrar um caracter ASCII, na próxima linha exiba no monitor uma mensagem apresentando o valor binário deste código e numa segunda linha exiba outra mensagem que apresente o número de bits "1" existentes no código ASCII.

3) Escreva um programa que peça ao usuário para entrar um caracter ASCII, na próxima linha exiba no monitor o caracter lido e numa segunda linha exiba outra mensagem que apresente o valor hexadecimal correspondente ao código ASCII. Repita este procedimento até que o usuário entre com *carriage return* CR. Exemplo:

```
Digite um caracter: Z  
O codigo ASCII de Z em hexa vale: 5Ah  
Digite um caracter: ...
```

4) Escreva um programa que peça ao usuário para entrar um número hexadecimal de 4 dígitos ou menos, terminado com CR, e exiba na próxima linha do monitor o mesmo número expandido em binário. Na leitura, faça com que o apenas letras maiúsculas de A a F sejam aceitas. Se houver um caracter

ilegal, o programa deve emitir uma mensagem instruindo o usuário a tentar novamente.



