Assembly Language for Intel-Based Computers, 5th Edition

Kip R. Irvine

Capítulo 7: Aritmética de Inteiros

Slides prepared by the author

Revision date: June 4, 2006

Índice

- **Instruções de Shift e Rotate**
- Aplicações de Shift e Rotate
- Instruções de Multiplicação e Divisão
- Adição e subtração estendida

Irvine, Kip R. Assembly Language for Intel-Based Computers 5/e, 2007.

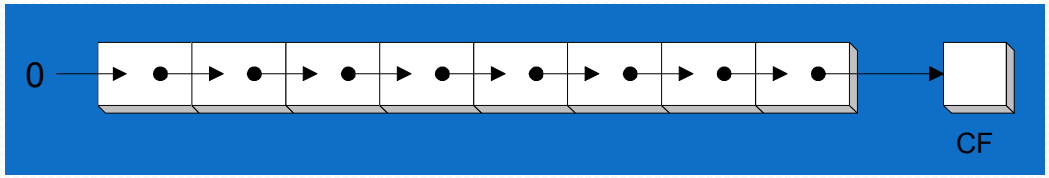
Instruções de Shift e Rotate

- Shift Lógico vs Aritmético
- Instrução SHL
- Instrução SHR
- Instruções SAL e SAR
- Instrução ROL
- Instrução ROR
- Instruções RCL e RCR
- Instruções SHLD/SHRD

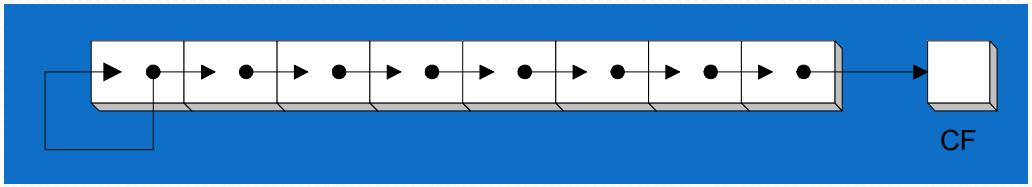
Irvine, Kip R. Assembly Language for Intel-Based Computers 5/e, 2007.

Shift lógico vs. aritmético

- O shift lógico preenche as lacunas criadas com zero:



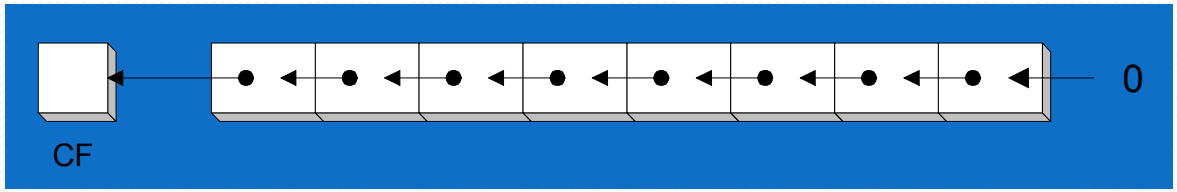
- O shift aritmético preenche as lacunas criadas com a cópia do bit de sinal:



Irvine, Kip R. Assembly Language for Intel-Based Computers 5/e, 2007.

Instrução SHL

- A instrução SHL (shift left) faz o deslocamento lógico à esquerda do operando destino, preenchendo o bit à direita com 0.



- Tipos de operando para SHL:

```
SHL reg,imm8
SHL mem,imm8
SHL reg,CL
SHL mem,CL
```

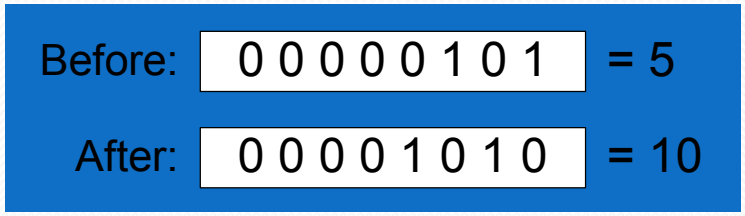
(mesmos para todas as instruções shift e rotate)

Irvine, Kip R. Assembly Language for Intel-Based Computers 5/e, 2007.

Fast Multiplication

Deslocando 1 bit à esquerda multiplica um número por 2

```
mov dl,5
shl dl,1
```



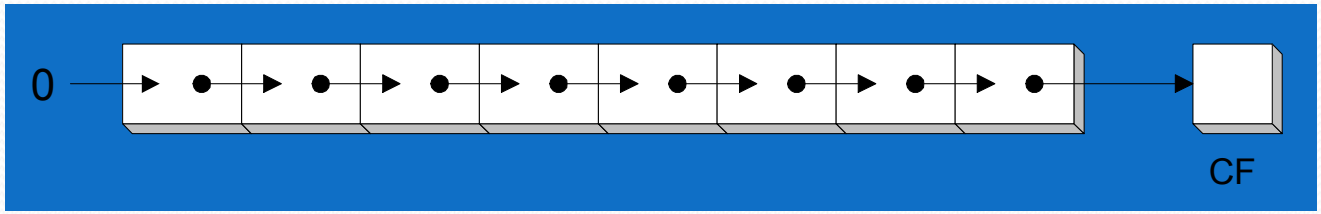
Deslocando à esquerda n bits multiplica o operando por 2^n

Exemplo, $5 * 2^2 = 20$

```
mov dl,5
shl dl,2                          ; DL = 20
```

Instrução SHR

- A instrução SHR (shift right) faz o deslocamento lógico à direita do operando destino. A posição do bit mais significativo é preenchido com zero.



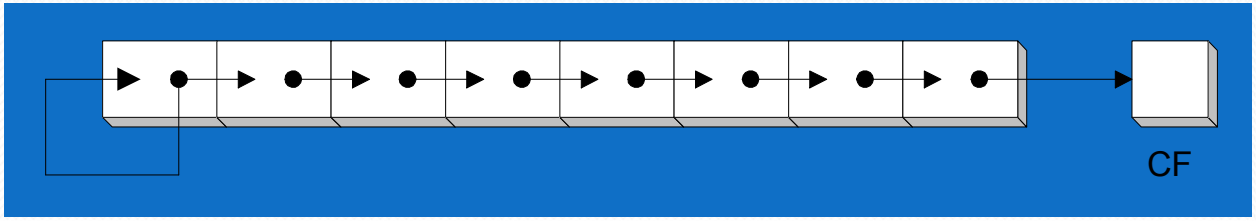
Deslocando *n* bits à direita divide o operando por 2^n

```
mov dl,80
shr dl,1                    ; DL = 40
shr dl,2                    ; DL = 10
```

Irvine, Kip R. Assembly Language for Intel-Based Computers 5/e, 2007.

Instruções SAL e SAR

- SAL (shift arithmetic left) é idêntico a SHL.
- SAR (shift arithmetic right) faz um deslocamento aritmético à direita no operando destino.



Um shift aritmético preserva o sinal do número.

```
mov dl,-80
sar dl,1                    ; DL = -40
sar dl,2                    ; DL = -10
```

Irvine, Kip R. Assembly Language for Intel-Based Computers 5/e, 2007.

Sua vez . . .

Indicar o valor em hexadecimal de AL após cada shift:

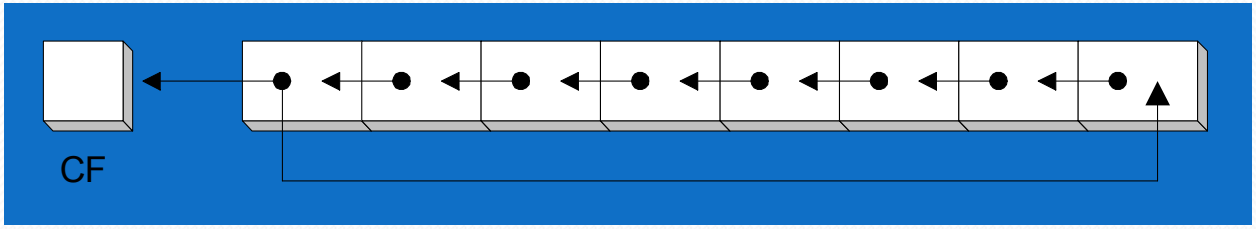
```
mov al,6Bh
shr al,1
shl al,3
mov al,8Ch
sar al,1
sar al,3
```

a. 35h
b. A8h
c. C6h
d. F8h

Irvine, Kip R. Assembly Language for Intel-Based Computers 5/e, 2007.

Instrução ROL

- ROL (rotate) desloca cada bit à esquerda
- O bit mais significativo é copiado no flag Carry e no bit menos significativo
- Nenhum bit é perdido



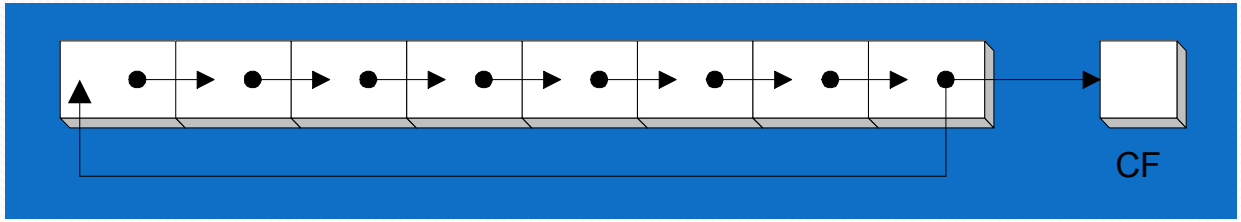

```
mov al,11110000b
rol al,1                    ; AL = 11100001b

mov dl,3Fh
rol dl,4                    ; DL = F3h
```

Irvine, Kip R. Assembly Language for Intel-Based Computers 5/e, 2007.

Instrução ROR

- ROR (rotate right) desloca cada bit à direita
- O bit menos significativo é copiado no flag Carry e na posição do bit mais significativo
- Nenhum bit é perdido




```
mov al,11110000b
ror al,1                    ; AL = 01111000b

mov dl,3Fh
ror dl,4                    ; DL = F3h
```

Irvine, Kip R. Assembly Language for Intel-Based Computers 5/e, 2007.

Sua vez . . .

Indicar o valor hexadecimal de AL após cada rotação:

```
mov al,6Bh
ror al,1
rol al,3
```

a. B5h

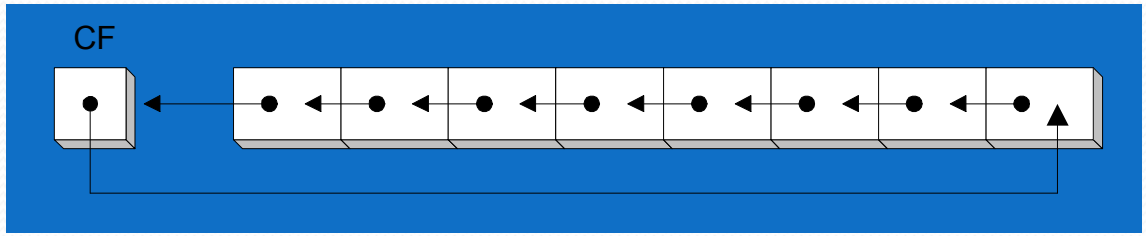
b. ADh

Irvine, Kip R. Assembly Language for Intel-Based Computers 5/e, 2007.

Instrução RCL

- RCL (rotate carry left) desloca cada bit à esquerda
- Copia o Carry flag para a posição menos significativa
- Copia o bit mais significativo no flag Carry

clc = clear carry:
carry=0

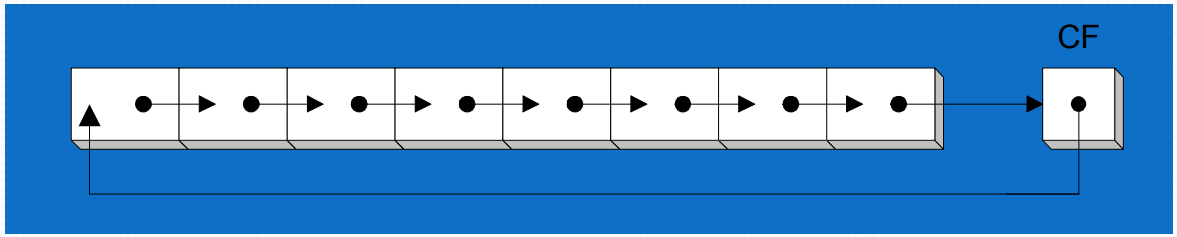



```
clc                         ; CF = 0
mov bl,88h                  ; CF,BL = 0 10001000b
rcl bl,1                    ; CF,BL = 1 00010000b
rcl bl,1                    ; CF,BL = 0 00100001b
```

Irvine, Kip R. Assembly Language for Intel-Based Computers 5/e, 2007.

Instrução RCR

- RCR (rotate carry right) desloca cada bit à direita
- Copia o flag Carry na posição mais significativa
- Copia o bit menos significativo no flag Carry



stc = set carry:
carry=1

```
stc                     ; CF = 1
mov ah,10h              ; CF,AH = 1 00010000b
rcr ah,1                ; CF,AH = 0 10001000b
```

Irvine, Kip R. Assembly Language for Intel-Based Computers 5/e, 2007.

Sua vez . . .

Indicar o valor hexadecimal de AL após cada rotação:

```
stc
mov al,6Bh
rcr al,1
rcl al,3
```

a. **B5h**

b. **AEh**

Irvine, Kip R. Assembly Language for Intel-Based Computers 5/e, 2007.

Instrução SHLD

- Desloca o operando destino um dado número de bits à esquerda
- As posições vazias resultantes são preenchidas pelos bits mais significativos do operando fonte
- O operando fonte não é afetado
- Sintaxe:

  SHLD *destination, source, count*

- Tipos de operando:

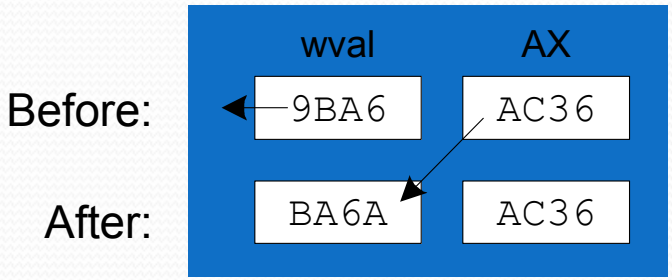
```
SHLD reg16/32, reg16/32, imm8/CL
SHLD mem16/32, reg16/32, imm8/CL
```

Irvine, Kip R. Assembly Language for Intel-Based Computers 5/e, 2007.

Exemplo de SHLD

Desloca wval 4 bits à esquerda e substitui os 4 bits menos significativos com os 4 bits mais significativos de AX:

```
.data
wval WORD 9BA6h
.code
mov  ax,0AC36h
shld wval,ax,4
```



Irvine, Kip R. Assembly Language for Intel-Based Computers 5/e, 2007.

Instrução SHRD

- Desloca o operando destino um dado número de bits à direita
- As posições vazias resultantes são preenchidas com os bits menos significativos do operando fonte
- O operando fonte não é afetado
- Sintaxe:

  SHRD *destination, source, count*

- Tipos de operando:

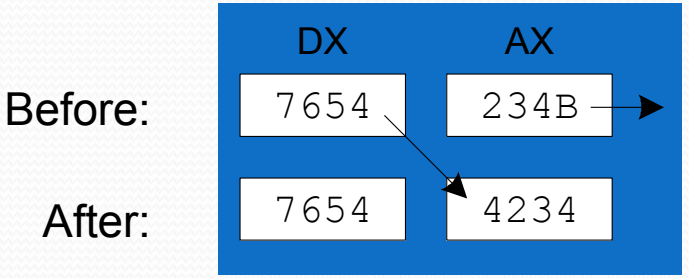
```
SHRD reg16/32, reg16/32, imm8/CL
SHRD mem16/32, reg16/32, imm8/CL
```

Irvine, Kip R. Assembly Language for Intel-Based Computers 5/e, 2007.

Exemplo de SHRD

Desloca AX 4 bits à direita e substitui os 4 bits mais significativos com os 4 bits menos significativos de DX:

```
mov   ax,234Bh
mov   dx,7654h
shrd  ax,dx,4
```



Irvine, Kip R. Assembly Language for Intel-Based Computers 5/e, 2007.

Sua vez . . .

Indicar em valor hexadecimal cada operando destino:

```
mov  ax,7C36h
mov  dx,9FA6h
shld dx,ax,4          ; DX = FA67h
shrd dx,ax,8          ; DX = 36FAh
```

Irvine, Kip R. Assembly Language for Intel-Based Computers 5/e, 2007.

Próxima seção

- Instruções de Shift e Rotate
- Aplicações de Shift e Rotate
- Instruções de Multiplicação e Divisão
- Adição e subtração estendida

Aplicações de Shift e Rotate

- Deslocando Doublewords múltiplos
- Multiplicação binária
- Mostrando bits binários
- Isolando uma cadeia de bits

Irvine, Kip R. Assembly Language for Intel-Based Computers 5/e, 2007.

Deslocando Doublewords múltiplos

- Os programas às vezes precisam deslocar todos os bits de um vetor, como o movimento de uma imagem gráfica de uma posição da tela para outra.
- O seguinte programa desloca um vetor de 3 doublewords 1 bit à direita:

```
.data
ArraySize = 3
array DWORD ArraySize DUP(99999999h)        ; 1001 1001...
.code
mov esi,0
shr array[esi],1              ; high dword
rcr array[esi + 4],1          ; middle dword, include Carry
rcr array[esi + 8],1          ; low dword, include Carry
```

Dica: não se esqueça que na memória
cada dword do array está em Little Endian

Multiplicação binária

- Sabemos que SHL faz a multiplicação sem sinal quando o multiplicador é potência de 2.
- É possível fatorar qualquer número binário em potência de 2.
  - Por exemplo, para multiplicar EAX * 36, fatorar 36 em 32 + 4 e usar a propriedade distributiva de multiplicação :

```
EAX * 36
= EAX * (32 + 4)
= (EAX * 32)+(EAX * 4)
```

```
mov eax,123
mov ebx,eax
shl eax,5          ; mult by 2^5
shl ebx,2          ; mult by 2^2
add eax,ebx
```

Sua vez . . .

Multiplicar AX por 26, usando deslocamento e adição.

Dica: 26 = 16 + 8 + 2.

```
mov ax,2                    ; test value

mov dx,ax
shl dx,4                    ; AX * 16
push dx                     ; save for later
mov dx,ax
shl dx,3                    ; AX * 8
shl ax,1                    ; AX * 2
add ax,dx                   ; AX * 10
pop dx                      ; recall AX * 16
add ax,dx                   ; AX * 26
```

Mostrando bits binários de eax

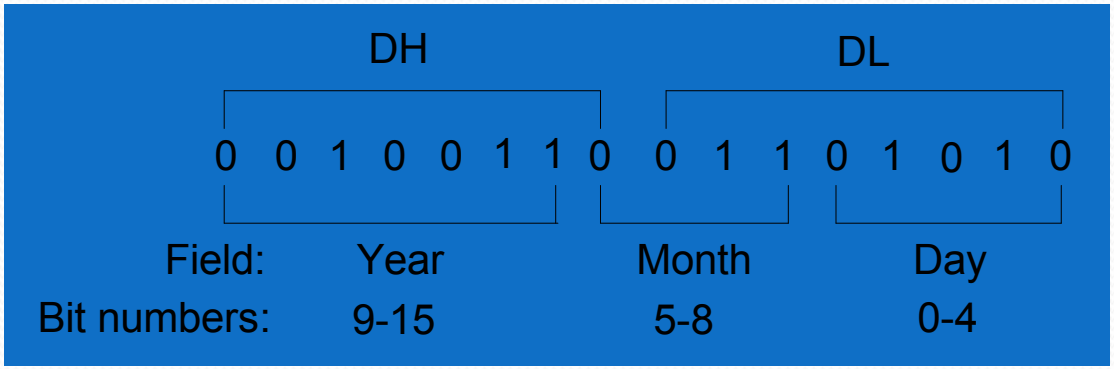
Algoritmo: deslocar o MSB para o flag Carry; se CF = 1, anexar o caractere "1“ à cadeia; caso contrário, anexar o caractere “0” . Repetir em loop, 32 vezes.

```
.data
buffer BYTE 32 DUP(0),0
.code
    mov ecx,32
    mov esi,OFFSET buffer
L1: shl eax,1
    mov BYTE PTR [esi],'0'
    jnc L2
    mov BYTE PTR [esi],'1'
L2: inc esi
    loop L1
```

Irvine, Kip R. Assembly Language for Intel-Based Computers 5/e, 2007.

Isolando uma cadeia de bits

- O campo de data do arquivo MS-DOS empacota o ano, mês e dia em 16 bits:



Isolar o campo mês:

```
mov ax,dx                    ; make a copy of DX
shr ax,5                     ; shift right 5 bits
and al,00001111b             ; clear bits 4-7
mov month,al                 ; save in month variable
```

Irvine, Kip R. Assembly Language for Intel-Based Computers 5/e, 2007.

Próxima seção

- Instruções de Shift e Rotate
- Aplicações de Shift e Rotate
- Instruções de Multiplicação e Divisão
- Adição e subtração estendida

Intervalo ?

Irvine, Kip R. Assembly Language for Intel-Based Computers 5/e, 2007.

Instruções de Multiplicação e Divisão

- Instrução MUL
- Instrução IMUL
- Instrução DIV
- Divisão inteira com sinal
- Instruções CBW, CWD, CDQ
- Instrução IDIV
- Implementando expressões Aritméticas

Irvine, Kip R. Assembly Language for Intel-Based Computers 5/e, 2007.

Instrução MUL

- A instrução MUL (unsigned multiply) multiplica um operando de 8-, 16-, ou 32-bit por AL, AX, ou EAX.

- Os formatos são:

```
MUL r/m8
MUL r/m16
MUL r/m32
```

Operandos implícitos:

Multiplicand	Multiplier	Product
AL	*r/m8*	AX
AX	*r/m16*	DX:AX
EAX	*r/m32*	EDX:EAX

Irvine, Kip R. Assembly Language for Intel-Based Computers 5/e, 2007.

Exemplos de MUL

100h * 2000h, usando operandos de 16-bits:

```
.data
val1 WORD 2000h
val2 WORD 100h
.code
mov ax,val1
mul val2        ; DX:AX = 00200000h, CF=1
```

Os flag Carry e Overflow apenas indicam se a metade superior contem dígitos significativos.

É impossível ocorrer estouro no operando de destino.

12345h * 1000h, usando operandos de 32-bits:

```
mov eax,12345h
mov ebx,1000h
mul ebx         ; EDX:EAX = 0000000012345000h, CF=0
```

Irvine, Kip R. Assembly Language for Intel-Based Computers 5/e, 2007.

Sua vez . . .

Quais seriam os valores em hexadecimal de DX, AX, e flag Carry após a execução das instruções seguintes?

```
mov ax,1234h
mov bx,100h
mul bx
```

DX = 0012h, AX = 3400h, CF = 1

Irvine, Kip R. Assembly Language for Intel-Based Computers 5/e, 2007.

Sua vez. . .

Quais serão os valores em hexadecimal de EDX, EAX, e flag Carry após a execução das seguintes instruções?

```
mov eax,00128765h
mov ecx,10000h
mul ecx
```

EDX = 00000012h, EAX = 87650000h, CF = 1

Instrução IMUL

- IMUL (signed integer multiply) multiplica um operando com sinal de 8-, 16-, ou 32-bits por AL, AX, ou EAX
- Preserva o sinal do produto estendendo o sinal para o registrador destino da metade mais significativa

Exemplo: multiplicar 48 * 4, usando operandos de 8-bits :

```
mov  al,48
mov  bl,4
imul bl              ; AX = 00C0h, OF=1
```

OF=1 porque AH recebe bits significativos, não somente extensão de sinal de AL.

Flags Carry=0 e Overflow=0 quando:

r/m8 AL := sign-extend of AL to 16 bits
r/m16 AX := sign-extend of AX to 32 bits
r/m32 EDX:EAX := sign-extend of EAX to 32 bits

Irvine, Kip R. Assembly Language for Intel-Based Computers 5/e, 2007.

Exemplos de IMUL

Multiplicar 4,823,424 * –423:

```
mov eax,4823424
mov ebx,-423
imul ebx          ; EDX:EAX = FFFFFFFF86635D80h, OF=0
```

OF=0 porque EDX é somente extensão de sinal de EAX.

Sua vez . . .

Quais serão os valores hexadecimal de DX, AX, flag Carry e overflow flag após a execução das seguintes instruções?

```
mov ax,8760h
mov bx,100h
imul bx
```

DX = FF87h, AX = 6000h, CF=1, OF = 1

Instrução DIV (sem sinal)

- A instrução DIV faz a divisão de 8-bit, 16-bit, e 32-bits em inteiros sem sinal
- O divisor é o único operando explícito (registrador ou memória)
- Formatos:

```
DIV r/m8
DIV r/m16
DIV r/m32
```

Operandos Default:

Dividend	Divisor	Quotient	Remainder
AX	*r/m8*	AL	AH
DX:AX	*r/m16*	AX	DX
EDX:EAX	*r/m32*	EAX	EDX

Flags: overflow e carry: indefinidas

Divide Overflow: resultado da divisão não cabe no operando destino (quociente) – interrupção do sistema!

Exemplos de DIV

Dividir 8003h por 100h, usando operandos de 16-bits:

```
mov dx,0                ; clear dividend, high
mov ax,8003h            ; dividend, low
mov cx,100h             ; divisor
div cx                  ; AX = 0080h, DX = 3
```

Mesma divisão, usando operandos de 32-bits:

```
mov edx,0               ; clear dividend, high
mov eax,8003h           ; dividend, low
mov ecx,100h            ; divisor
div ecx                 ; EAX = 00000080h, EDX = 3
```

Sua vez . . .

Quais os valores em hexadecimal de DX e AX após a execução das seguintes instruções? Ou, indicar se ocorre divide overflow:

```
mov dx,0087h
mov ax,6000h
mov bx,100h
div bx
```

DX = 0000h, AX = 8760h

Sua vez . . .

Quais os valores em hexadecimal de DX e AX após a execução das seguintes instruções? Ou, indicar se ocorrer divide overflow:

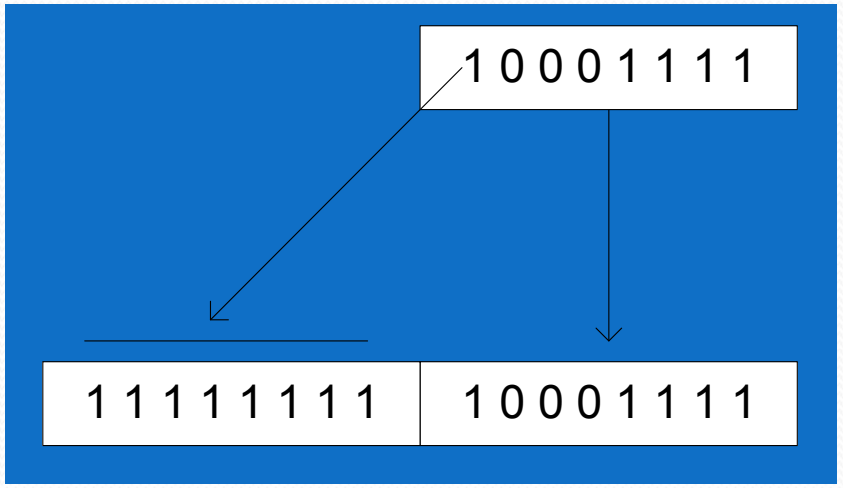
```
mov dx,0087h
mov ax,6002h
mov bx,10h
div bx
```

Divide Overflow

Irvine, Kip R. Assembly Language for Intel-Based Computers 5/e, 2007.

IDIV - Divisão inteira com sinal

- Os inteiros com sinal devem ser estendidos em sinal antes da divisão ser realizada
  - Preencher os byte/word/doubleword mais significativos com uma cópia do bit de sinal do byte/word/doubleword menos significativo
- Por exemplo, o byte mais significativo contem uma cópia do bit de sinal do byte menos significativo:



Flags: overflow e carry: indefinidas

Divide Overflow: resultado da divisão não cabe no operando destino (quociente) – interrupção do sistema!

Irvine, Kip R. Assembly Language for Intel-Based Computers 5/e, 2007.

Instruções CBW, CWD, CDQ

- As instruções CBW, CWD e CDQ realizam importantes operações de extensão de sinal:
  - CBW (convert byte to word) estende AL para AH
  - CWD (convert word to doubleword) estende AX para DX
  - CDQ (convert doubleword to quadword) estende EAX para EDX

- Exemplo:

```
mov eax,0FFFFFF9Bh ; (-101)
cdq                ; EDX:EAX = FFFFFFFFFFFFFF9Bh
```

Irvine, Kip R. Assembly Language for Intel-Based Computers 5/e, 2007.

Instrução IDIV

- IDIV faz a divisão de inteiro com sinal
- Mesma sintaxe e operandos como na instrução DIV

Exemplo: divisão de 8-bits de –48 por 5

```
mov  al,-48
cbw                ; extend AL into AH
mov  bl,5
idiv bl            ; AL = -9,  AH = -3
```

Irvine, Kip R. Assembly Language for Intel-Based Computers 5/e, 2007.

Exemplos de IDIV

Exemplo: divisão de 16-bits de –48 por 5

```
mov  ax,-48
cwd                 ; extend AX into DX
mov  bx,5
idiv bx             ; AX = -9,  DX = -3
```

Exemplo: divisão de 32-bits de –48 por 5

```
mov  eax,-48
cdq                 ; extend EAX into EDX
mov  ebx,5
idiv ebx            ; EAX = -9,  EDX = -3
```

Sua vez . . ..

Quais os valores em hexadecimal de DX e AX após a execução das seguintes instruções? Ou, se ocorrer divide overflow , indicar isso como resposta:

```
mov  ax,0FDFFh                ; -513
cwd
mov  bx,100h
idiv bx
```

DX = FFFFh (–1), AX = FFFEh (–2)

Expressões aritméticas sem sinal

- Algumas boas razões para aprender expressões de inteiros:
  - Aprender como compiladores as fazem
  - Testar o entendimento de MUL, IMUL, DIV, IDIV
  - Check de overflow (flags Carry e Overflow)

Exemplo: `var4 = (var1 + var2) * var3`

```
; Assume unsigned operands
mov  eax,var1
add  eax,var2          ; EAX = var1 + var2
mul  var3              ; EAX = EAX * var3
jc   TooBig            ; check for carry
mov  var4,eax          ; save product
```

Irvine, Kip R. Assembly Language for Intel-Based Computers 5/e, 2007.

Expressões aritmética com sinal (1 de 2)

Exemplo: `eax = (-var1 * var2) + var3`

```
mov  eax,var1
neg  eax
imul var2
jo   TooBig          ; check for overflow
add  eax,var3
jo   TooBig          ; check for overflow
```

Exemplo: `var4 = (var1 * 5) / (var2 - 3)`

```
mov  eax,var1          ; left side
mov  ebx,5
imul ebx               ; EDX:EAX = product
mov  ebx,var2          ; right side
sub  ebx,3
idiv ebx               ; EAX = quotient
mov  var4,eax
```

Irvine, Kip R. Assembly Language for Intel-Based Computers 5/e, 2007.

Expressões aritmética com sinal (2 de 2)

Exemplo : `var4 = (var1 * -5) / (-var2 % var3);`

```
mov  eax,var2          ; begin right side
neg  eax
cdq                    ; sign-extend dividend
idiv var3              ; EDX = remainder
mov  ebx,edx           ; EBX = right side
mov  eax,-5            ; begin left side
imul var1              ; EDX:EAX = left side
idiv ebx               ; final division
mov  var4,eax          ; quotient
```

As vezes é mais fácil calcular o termo à direita primeiro.

Irvine, Kip R. Assembly Language for Intel-Based Computers 5/e, 2007.

Sua vez . . .

Implementar a seguinte expressão usando inteiros de 32 bits com sinal:

```
eax = (ebx * 20) / ecx
```

```
mov eax,20
imul ebx
idiv ecx
```

Sua vez . . .

Implementar a seguinte expressão usando inteiros de 32 bits com sinal. Salvar e restaurar EDX:

```
eax = (ecx * edx) / eax
```

```
push   edx
push   eax                  ; EAX needed later
mov    eax,ecx
imul   edx                  ; left side: EDX:EAX
pop    ebx                  ; saved value of EAX
idiv   ebx                  ; EAX = quotient
pop    edx                  ; restore EDX
```

Irvine, Kip R. Assembly Language for Intel-Based Computers 5/e, 2007.

Sua vez . . .

Implementar a seguinte expressão usando inteiros de 32 bits com sinal. Não modificar nenhuma variável a não ser var3:

```
var3 = (var1 * -var2) / (var3 – ebx)
```

```
mov  eax,var1
mov  edx,var2
neg  edx
imul edx              ; left side: EDX:EAX
mov  ecx,var3
sub  ecx,ebx
idiv ecx              ; EAX = quotient
mov  var3,eax
```

Próxima seção

- Instruções de Shift e Rotate
- Aplicações de Shift e Rotate
- Instruções de Multiplicação e Divisão
- Adição e subtração estendida

Irvine, Kip R. Assembly Language for Intel-Based Computers 5/e, 2007.

52

Adição e subtração estendida

- Instrução ADC
- Extended Precision Addition
- Instrução SBB
- Extended Precision Subtraction

Irvine, Kip R. Assembly Language for Intel-Based Computers 5/e, 2007.

Adição com precisão estendida

- Adicionando 2 operandos que são maiores que o tamanho máximo da palavra (32 bits).
  - Virtualmente não deve existir limite para o tamanho dos operandos
- A aritmética deve ser realizada em etapas
  - O valor de Carry de uma etapa é passado para a próxima etapa.

Irvine, Kip R. Assembly Language for Intel-Based Computers 5/e, 2007.

Instrução ADC

- A instrução ADC soma o operando fonte e o flag de Carry ao operando destino.
- Operandos são valores binários
  - Mesma sintaxe do ADD, SUB, etc.
- Exemplo
- Somar dois inteiros de 32-bits (FFFFFFFFh + FFFFFFFFh), produzindo uma soma de 64-bit em EDX:EAX:

```
mov edx,0
mov eax,0FFFFFFFFh
add eax,0FFFFFFFFh
adc edx,0                    ;EDX:EAX = 00000001FFFFFFFEh
```

Irvine, Kip R. Assembly Language for Intel-Based Computers 5/e, 2007.

Exemplo de adição com precisão estendida

- Tarefa: somar 1 a EDX:EAX
  - Valor inicial de EDX:EAX: 00000000FFFFFFFFh
  - Somar os 32 bits menos significativos primeiro, acionando o flag Carry.
  - Somar os 32 bits mais significativos, e incluir o flag Carry.

```
mov edx,0              ; set upper half
mov eax,0FFFFFFFFh     ; set lower half
add eax,1              ; add lower half
adc edx,0              ; add upper half

  EDX:EAX = 00000001 00000000
```

Instrução SBB

- A instrução SBB subtrai o operando fonte e o flag Carry do operando destino.
- sintaxe:
  - Mesmo que a instrução ADC

Exemplo de subtração estendida

- Tarefa: Subtrair 1 de EDX:EAX
  - Valor inicial de EDX:EAX: 0000000100000000h
  - Subtrair os 32 bits menos significativos primeiro, acionando o flag Carry.
  - Subtrair os 32 bits mais significativos, incluindo o flag Carry.

```
mov edx,1              ; set upper half
mov eax,0              ; set lower half
sub eax,1              ; subtract lower half
sbb edx,0              ; subtract upper half

EDX:EAX = 00000000 FFFFFFFF
```

Sumário

- Instruções Shift e rotate são algumas das melhores ferramentas da linguagem assembly
  - Controle mais fino que em linguagens de alto nível
  - SHL, SHR, SAR, ROL, ROR, RCL, RCR
- MUL e DIV – operações inteiras
  - Próximas de SHL e SHR
  - CBW, CDQ, CWD: preparação para divisão
- Aritmética de precisão estendida: ADC, SBB

Fim