

## 8. Instruções de multiplicação e divisão

### 8.1 Instruções de multiplicação

**MUL fonte**

**IMUL fonte**

- MUL (*multiply*) -> usada com números em representação não-sinalizada
- IMUL (*integer multiply*) -> usada com números sinalizados
- **Multiplicação com números em formato *byte*:**
  - registradores de 8 bits e variáveis de tipo DB
  - segundo operando é assumido em AL
  - resultado (destino) pode atingir 16 bits e se encontra em AX
- **Multiplicação com números em formato *word*:**
  - registradores de 16 bits e variáveis de tipo DW
  - segundo operando é assumido em AX
  - resultado pode atingir 32 bits (tamanho *doubleword*) e se encontra em
    - DX -> 16 bits mais significativos (*high word*)
    - AX -> 16 bits menos significativos (*low word*)
- Para **números positivos** (MSB = 0), MUL e IMUL dão o mesmo resultado.
- **Flags afetados:**

SF, ZF, AF, PF -> indefinidos

após MUL, CF/OF (ambos) = 0 , se a metade superior do resultado é 0  
= 1 , caso contrário

após IMUL, CF/OF (ambos) = 0 , se a metade superior do resultado for  
extensão do sinal da metade inferior  
= 1 , caso contrário

**Exemplos de casos de multiplicação:**

1) Suponha que AX contenha 0001h BX contenha FFFFh:

antes: AX = 0001 h = 0000 0000 0000 0001b = 1 ou +1  
 BX = FFFF h = 1111 1111 1111 1111b = 65535 ou -1

Instrução	Resultado decimal	Resultado hexadecimal	DX	AX	CF/OF
MUL BX	65535	0000 FFFF h	0000 h	FFFF h	0
IMUL BX	-1	FFFF FFFF h	FFFF h	FFFF h	0

2) Suponha que AX contenha 0FFF h:

antes: AX = 0FFF h = 0000 1111 1111 1111 h = 4095 ou + 4095

Instrução	Resultado decimal	Resultado hexadecimal	DX	AX	CF/OF
MUL AX	16769025	00FF E001 h	00FF h	E001 h	1
IMUL AX	16769025	00FF E001 h	00FF h	E001 h	1

3) Suponha que AL contenha 80h BL contenha FFh:

antes: AL = 80 h = 1000 0000 b = 128 ou -128  
 BL = FF h = 1111 1111 b = 255 ou -1

Instrução	Resultado decimal	Resultado hexadecimal	AH	AL	CF/OF
MUL BL	32640	7F80 h	7F h	80 h	1
IMUL BL	128	0080 h	00 h	80 h	1

## 8.2 Instruções de divisão

**DIV fonte**  
**IDIV fonte**

- **DIV** (*divide*) -> usada com números em representação não-sinalizada
- **IDIV** (*integer divide*) -> usada com números sinalizados
- **fonte** deve ser considerado como **divisor** (não pode ser uma constante)

### Divisão com números em formato *byte*:

- o divisor é assumido ser de 8 bits (1 byte)
- o dividendo é assumido estar em AX (16 bits)
- após a execução: o quociente de 8 bits estará em AL  
o resto de 8 bits estará em AH

### Divisão com números em formato *word*:

- o divisor é assumido ser de 16 bits (1 word)
- o dividendo é assumido ser de 32 bits:
  - DX -> 16 bits mais significativos do dividendo (*high word*)
  - AX -> 16 bits menos significativos do dividendo (*low word*)
- após a execução: o quociente de 16 bits estará em AX  
o resto de 16 bits estará em DX

- Para **números positivos** (MSB = 0), DIV e IDIV fornecem o mesmo resultado.
- **Flags afetados:** todos ficam indefinidos
- Em divisão de números em representação sinalizada, o **resto** possui o mesmo sinal do **dividendo**.

### Exemplos de casos de divisão:

1) Suponha que DX e AX contenham 0000h e 0005h, e BX contenha FFFEh:

antes:     DX : AX   = 0000 0005 h =       5   ou   + 5  
          BX        =       FFFE h = 65534   ou   - 2

Instrução	Quociente decimal	Resto decimal	AX	DX
DIV BX	0	5	0000 h	0005 h
IDIV BX	-2	1	FFFE h	0001 h

2) Suponha que AX contenha 0005 h e BL contenha FF h:

antes:     AX = 0005 h   =       5   ou   + 5  
          BL =   FF h    =       256   ou   - 1

Instrução	Quociente decimal	Resto decimal	AL	AH
DIV BL	0	5	00 h	05 h
IDIV BL	- 5	0	FB h	00 h

3) Suponha que AX contenha 00FB h e BL contenha FF h:

antes:     AX = 00FB h   =       251   ou   + 251  
          BL =   FF h    =       256   ou   - 1

Instrução	Quociente decimal	Resto decimal	AL	AH
DIV BL	0	251	00 h	FB h
IDIV BL	-251 *	-	-	-

(\*) como -251 não cabe em AL (8 bits) -> ocorre ***DIVIDE OVERFLOW*** situação de **erro catastrófico** que faz com que o **programa termine**.

## 8.3 Extensão do sinal do dividendo

### a) Em operações em formato word:

Caso o dividendo de uma divisão (composto de **DX : AX**) ocupe apenas **AX**, **DX** deve ser preparado, pois é sempre considerado:

- Em **DIV** -> **DX** deve ser **zerado**
- EM **IDIV** -> **DX** deve ter a **extensão de sinal** de AX

### CWD

- instrução sem operandos (zero operandos) que converte **word** para **doubleword** e estende o sinal de AX para DX;
- deve ser usada com **IDIV**

### b) Em operações em formato byte:

Caso o dividendo de uma divisão (composto por **AX**) ocupe apenas **AL**, **AH** deve ser preparado, pois é sempre considerado.

- Em **DIV** -> **AH** deve ser **zerado**
- EM **IDIV** -> **AH** deve ter a **extensão de sinal** de AL

### CBW

- instrução sem operandos (zero operandos) que converte **byte** para **word** e estende o sinal de AL para AH;
- deve ser usada com **IDIV**

## Exemplos de divisões com ajuste de extensão:

1) Crie um trecho de programa que divida -1250 por 7.

```

...
MOV AX, -1250    ;AX recebe o dividendo
CWD              ;estende o sinal de AX para DX
MOV BX,7         ;BX recebe o divisor
IDIV BX         ;executa a divisão
                ;após a execução, AX recebe o
                ;quociente e DX recebe o resto
...

```

2) Crie um trecho de programa que divida a variável sinalizada XBYTE por -7.

```

...
MOV AL,XBYTE    ;AL recebe o dividendo
CBW             ;estende o sinal (eventual) de
                ;AL para AH
MOV BL, -7      ;BL recebe o divisor
IDIV BL        ;executa a divisão
                ;após a execução, AL recebe o
                ;quociente e AH recebe o resto
...

```

**Obs:** Não há efeito de **CBW** e **CWD** sobre os **FLAGS**.

## 8.4 E/S de números decimais

### • Entrada de números decimais:

- *string* de caracteres números de 0 a 9, fornecidos pelo teclado;
- CR é o marcador de fim de *string*;
- AX é assumido como registrador de armazenamento;
- valores decimais permitidos na faixa de - 32768 a + 32767;
- sinal negativo deve ser apresentado.

### Algoritmo básico em linguagem de alto nível:

```

total = 0
negativo = FALSO
ler um caracter
CASE caracter OF
    ' - ' : negativo = VERDADEIRO e ler um caracter
    ' + ' : ler um caracter
END_CASE
REPEAT
    converter caracter em valor binário
    total = 10 x total + valor binário
    ler um caracter
UNTIL caracter é um carriage return (CR)
IF negativo = VERDADEIRO
    THEN total = - (total)
END_IF

```

Obs: o *loop* do tipo **CASE** pode ser entendido como um **IF múltiplo**, que testa simultaneamente os vários "casos": se algum deles for verdadeiro, executa as instruções relacionadas; se todos os "casos" forem falsos, não executa nada e vai para o fim do CASE.

**Subrotina de entrada de números decimais em Linguagem Montadora:**

```

ENTDEC PROC    ;le um numero decimal da faixa de -32768 a +32767
;variaveis de entrada: nenhuma (entrada de digitos pelo teclado)
;variaveis de saida: AX -> valor binario equivalente do numero decimal
    PUSH BX
    PUSH CX
    PUSH DX    ;salvando registradores que serão usados
    XOR BX,BX  ;BX acumula o total, valor inicial 0
    XOR CX,CX  ;CX indicador de sinal (negativo = 1), inicial = 0
    MOV AH,1h
    INT 21h    ;le character no AL
    CMP AL, '-' ;sinal negativo?
    JE MENOS
    CMP AL, '+' ;sinal positivo?
    JE MAIS
    JMP NUM    ;se nao é sinal, então vá processar o character
MENOS: MOV CX,1 ;negativo = verdadeiro
MAIS:  INT 21h ;le um outro character
NUM:   AND AX,000Fh ;junta AH a AL, converte character para binário
    PUSH AX    ;salva AX (valor binário) na pilha
    MOV AX,10  ;prepara constante 10
    MUL BX    ;AX = 10 x total, total está em BX
    POP BX    ;retira da pilha o valor salvo, vai para BX
    ADD BX,AX ;total = total x 10 + valor binário
    MOV AH,1h
    INT 21h   ;le um character
    CMP AL,0Dh ;é o CR ?
    JNE NUM   ;se não, vai processar outro dígito em NUM
    MOV AX,BX ;se é CR, então coloca o total calculado em AX
    CMP CX,1  ;o numero é negativo?
    JNE SAIDA ;não
    NEG AX    ;sim, faz-se seu complemento de 2
SAIDA: POP DX
    POP CX
    POP BX    ;restaura os conteúdos originais
    RET      ;retorna a rotina que chamou
ENTDEC ENDP

```



**- Saída de números decimais:**

- AX é assumido como registrador de armazenamento;
- valores decimais na faixa de - 32768 a + 32767;
- exibe sinal negativo, se o conteúdo de AX for negativo;
- *string* de caracteres números de 0 a 9, exibidos no monitor de vídeo.

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

```

IF AX < 0
  THEN      exibe um sinal de menos
            substitui-se AX pelo seu complemento de 2
END_IF
contador = 0
REPEAT
  dividir quociente por 10
  colocar o resto na pilha
  contador = contador + 1
UNTIL quociente = 0
FOR contador vezes DO
  retirar um resto (número) da pilha
  converter para caracter ASCII
  exibir o caracter no monitor
END_FOR
  
```

**Idéia básica da técnica de decomposição decimal do número em AX:**

			Pilha	
24618	dividido por 10 = 2461	com resto <b>8</b> ->	0008h	
2461	dividido por 10 = 246	com resto <b>1</b> ->	0001h	
246	dividido por 10 = 24	com resto <b>6</b> ->	0006h	
24	dividido por 10 = 2	com resto <b>4</b> ->	0004h	
2	dividido por 10 = <b>0</b>	com resto <b>2</b> ->	0002h	<b>&lt;- Topo</b>

**Subrotina de saída de números decimais em Linguagem Montadora:**

```

SAIDEC PROC ;exibe o conteúdo de AX como decimal inteiro com sinal
;variáveis de entrada: AX -> valor binário equivalente do número decimal
;variáveis de saída: nenhuma (exibição de dígitos direto no monitor de vídeo)
    PUSH AX
    PUSH BX
    PUSH CX
    PUSH DX ;salva na pilha os registradores usados
    OR AX,AX ;prepara comparação de sinal
    JGE PT1 ;se AX maior ou igual a 0, vai para PT1
    PUSH AX ;como AX menor que 0, salva o número na pilha
    MOV DL,' - ' ;prepara o caractere ' - ' para sair
    MOV AH,2h ;prepara exibição
    INT 21h ;exibe ' - '
    POP AX ;recupera o número
    NEG AX ;troca o sinal de AX (AX = - AX)
;obtido dígitos decimais e salvando-os temporariamente na pilha
PT1: XOR CX,CX ;inicializa CX como contador de dígitos
    MOV BX,10 ;BX possui o divisor
PT2: XOR DX,DX ;inicializa o byte alto do dividendo em 0; restante é AX
    DIV BX ;após a execução, AX = quociente; DX = resto
    PUSH DX ;salva o primeiro dígito decimal na pilha (1o. resto)
    INC CX ;contador = contador + 1
    OR AX,AX ;quociente = 0 ? (teste de parada)
    JNE PT2 ;não, continuamos a repetir o laço
;exibindo os dígitos decimais (restos) no monitor, na ordem inversa
    MOV AH,2h ;sim, termina o processo, prepara exibição dos restos
PT3: POP DX ;recupera dígito da pilha colocando-o em DL (DH = 0)
    ADD DL,30h ;converte valor binário do dígito para caractere ASCII
    INT 21h ;exibe caractere
    LOOP PT3 ;realiza o loop até que CX = 0
    POP DX ;restaura o conteúdo dos registros
    POP CX
    POP BX
    POP AX ;restaura os conteúdos dos registradores
    RET ;retorna à rotina que chamou
SAIDEC ENDP

```

## Exemplo de um programa utilizando ENTDEC e SAIDEC

```

TITLE PROGRAMA DE E/S DECIMAL
.MODEL SMALL
.STACK 100h
.DATA
MSG1 DB 'Entre um numero decimal na faixa de -32768 a 32767:$'
MSG2 DB 0Dh, 0Ah, 'Confirmando a entrada efetuada, voce digitou:$'
.CODE
PRINCIPAL PROC
    MOV AX,@DATA
    MOV DS,AX
    MOV AH,9h
    LEA DX,MSG1
    INT 21h
;entrada do número
    CALL ENTDEC          ;chama subrotina ENTDEC
    PUSH AX              ;salva temporariamente o número na pilha
;exibindo a segunda mensagem
    MOV AH,9h
    LEA DX,MSG2
    INT 21h              ;exibe a segunda mensagem
;exibindo o número lido
    POP AX               ;recupera o número na pilha
    CALL SAIDEC          ;chama subrotina SAIDEC
;saida para o DOS
    MOV AH,4Ch
    INT 21h
PRNCIPAL ENDP
INCLUDE: C:\<diretorio_de_trabalho>\ENTDEC.ASM
INCLUDE: C:\<diretorio_de_trabalho>\SAIDEC.ASM
END PRINCIPAL

```

**Exercícios sugeridos:**

1) Dê os conteúdos de DX, AX e CF/OF após a execução de cada uma das seguintes instruções (operandos de 16 bits com resultado em 32 bits):

a) MUL BX ;se AX = 0008h e BX = 0003h

b) IMUL CX ;se AX = 0005h e CX = FFFFh

2) Dê os conteúdos de AX e CF/OF após a execução de cada uma das seguintes instruções (operandos de 8 bits com resultado em 16 bits):

a) MUL BL ;se AL = ABh e BL = 10h

b) IMUL BYTE1 ;se AL = 02h e BYTE1 = FBh

3) Dê os conteúdos de AX e DX após a execução de cada uma das seguintes instruções (dividendo de 32 bits, divisor de 16 bits, resultando em quociente e resto ambos de 16 bits):

a) DIV BX ;se DX = 0000h, AX = 0007h e BX = 0002h

b) IDIV BX ;se DX = FFFFh, AX = FFFCh e BX = 0003h

4) Dê os conteúdos de AL e AH após a execução de cada uma das seguintes instruções (dividendo de 16 bits, divisor de 8 bits, resultando em quociente e resto ambos de 8 bits):

a) DIV BL ;se AX = 000Dh e BL = 03h

b) IDIV BL ;se AX = FFFBh e BL = FEh

5) O que ocorre após a execução de:

a) CWD ;se AX = 8ABCh

b) CBW ;se AL = 5Fh

6) Modifique a subrotina ENTDEC para que a mesma verifique se o dígito decimal que entra pelo teclado está na faixa de 0 a 9, e se o número final produziu ou não **overflow**.

Dica: No algoritmo apresentado, pode ocorrer **overflow** em dois momentos:

- quando  $AX = 10 \times \text{total}$
- quando  $\text{total} = 10 \times \text{total} + \text{valor binário}$

7) Escreva um programa que leia uma quantidade de tempo, expressa em segundos, menor ou igual a 65535, e apresente esta quantidade em horas, minutos e segundos (formato HH:MM:SS). Apresente mensagens adequadas e utilize as subrotinas ENTDEC e SAIDEC para realizar a entrada e saída de dígitos decimais (será necessário adaptá-las).

8) Escreva uma subrotina que calcule  $X^n$ , enésima potência de X, onde as variáveis X e n estão respectivamente em AX e BX. O resultado é passado de volta para a rotina que chamou em dois registradores: DX (16 bits mais significativos) e AX (16 bits menos significativos). Inclua um teste de **overflow** caso o resultado não caiba em 32 bits.

9) Escreva um programa completo que peça ao usuário, mediante mensagens adequadas, para entrar um número X, entrar um expoente n e apresentar a enésima potência de X. Utilize a subrotina desenvolvida no exercício 8 para o cálculo de  $X^n$ . Utilize as subrotinas ENTDEC e SAIDEC para E/S de números decimais. Apresente uma mensagem adequada em caso de **overflow**.