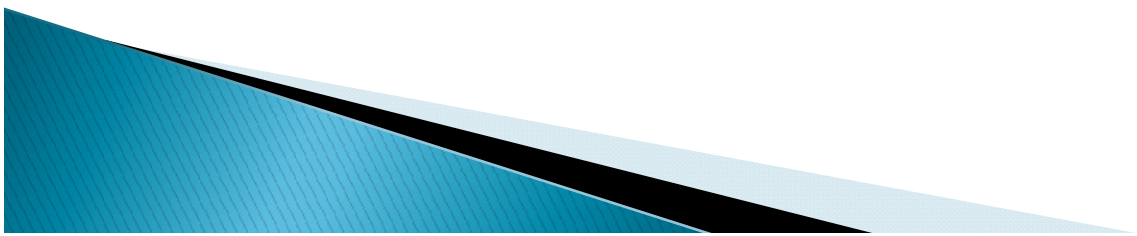
CHAPTER 9
Multiplication and Division Instructions



Signed Vs Unsigned Multiplication

- Signed and Unsigned numbers must be interpreted accordingly
- P = 10000000 * 11111111
- Unsigned Interpretation
  - P = 128 * 255 = 32640
- Signed Interpretation
  - P = -128 * -1 = 128
- Hence
  - P = 0111111110000000b ; unsigned multiplication
  - P = 0000000010000000b ; signed multiplication
- PRODUCT IS DIFFERENT with respect to INTERPRETATION (for numbers that are negative)

Multiplication instructions

- imul *source*
  - Signed multiply [Integer MULtiply]
- mul *source*
  - Unsigned multiply
- **Byte and Word Multiplication** (A X B)
  - If two **bytes** are multiplied, the result is a 16-bit **word**
    - A: **source**
    - B: **AL**
    - product: **AX**
  - If two **words** are multiplied, the result is a 32-bit *doubleword*
    - A: **source**
    - B: **AX**
    - Product: DX:AX
      - Product (ms 16 bits): **DX**
      - Product (ls 16 bits): **AX**

Multiplication instructions

- *source* can be a **register** or **memory byte/word, but can not be a constant**
- Byte form
  - AX=AL**source*
- Word form
  - DX:AX=AX**source*
- CF/OF
  - MUL
    - 0: upper half of result is 0
  - IMUL
    - 0: if upper half is sign extension of lower half.
  - CF/OF = 1 means product is too big for lower half of the destination (AL for byte and AX for word)

SF, ZF, AF, and PF Undefined

•If ax contains 0001h and bx contains FFFFh

•mul bx; dx = 0000h ax = FFFFh

•imul bx ; dx = FFFFh ax = FFFFh (-1)

More Examples

•AX=FFFFh,BX=FFFFh

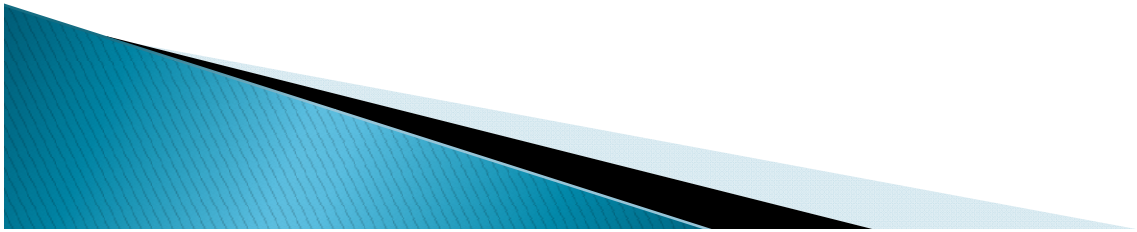
Instruction	Hex Product	DX	AX	CF/OF
MUL BX	FFFE0001 (4294836225)	FFFE (!zero)	0001	1
IMUL BX	1	0000	0001	0

•AX=80h,BX=FFh

Instruction	Hex Product	AH	AL	CF/OF
MUL BX	7F80 (128)	7F(!zero)	80	1
IMUL BX	0080	00 (no sign extension)	80	1

Division instructions

- **cbw**
  - convert byte to word
- **cwd**
  - convert word to doubleword
- **div *source***
  - unsigned divide
- **idiv *source***
  - signed divide

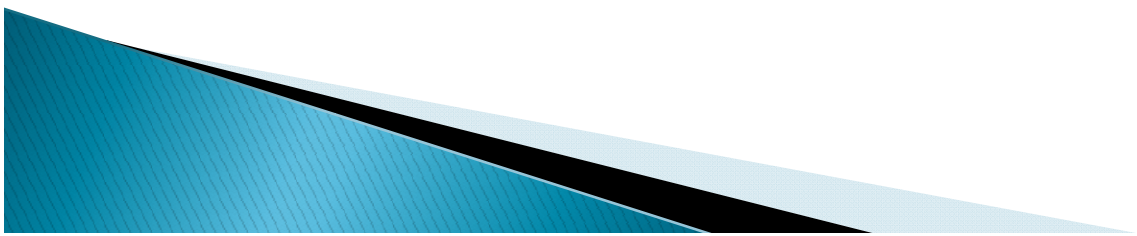


Byte and Word Division (A/B)

- When division is performed
  - two results: the quotient and the remainder
  - Quotient and remainder are the same **size** as the divisor
  - Divisor can not be a constant

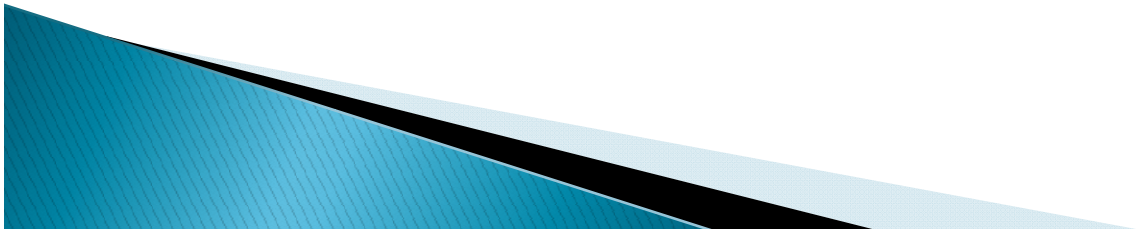
Byte and Word Division (A/B) (contd.)

- For the byte form,
  - Divisor, B: **source** ;Dividend , A: **AX**
  - Quotient : **AL** ;Remainder: **AH**
  - AL = AX / divisor; divisor is BYTE
  - AH = AX % divisor;
- For the word form,
  - Divisor, B: **source** ;Dividend , A: **DX:AX**
  - Quotient: **AX** ; Remainder: **DX**
  - AX = DX:AX / divisor ; divisor is WORD
  - DX = DX:AX % divisor



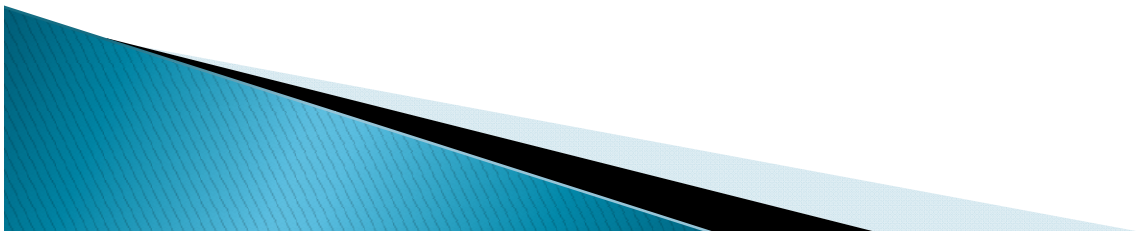
An Example

```
;If dx = 0000h, ax = 0005h, and
                    bx = FFFEh (-2)
div bx;   ax = 0000h    dx = 0005h
idiv bx;  ax = FFFEh    dx = 0001h
```



Divide Overflow

- It is possible that the **quotient** will be too big to fit in the specified destination (**al** or **ax**)
- if the **divisor** is much smaller than the **dividend**
- the program terminates and the system displays the message "**Divide Overflow**"



Sign Extension of the Dividend

- Word division
  - The dividend is in **dx:ax** even if the actual dividend will fit in **ax**
  - For **div**, **dx** should be cleared
  - For **idiv**, **dx** should be made the sign extension of **ax** using **cwd**

e.g. -1250/7

```
MOV AX,-1250
CWD ; sign extend
MOV BX,7
IDIV BX
```

Sign Extension of the Dividend

- Byte division
  - The dividend is in **ax** even if the actual dividend will fit in **al**
  - For **div**, **ah** should be cleared
  - For **idiv**, **ah** should be made the sign extension of **al** using **cbw**

Thank You