Assembly : Arithmetic & Logic instructions

Author	Md. Zarif Ul Alam
Date Created	@March 16, 2021 7:06 PM
Tag	CSE315 Microprocessors Microcontrollers and Embedded Systems

AND

```
; AND destination , source

AND AL , 7FH ; 7FH is 0111 1111 , so sign bit is cleared
```

OR

```
; OR destination , source

OR AL , 81H ; 81H is 1000 0001 , so MSB and LSB is set to 1
```

NOT

```
; NOT destination

NOT AX ; whole content is bit flipped
```

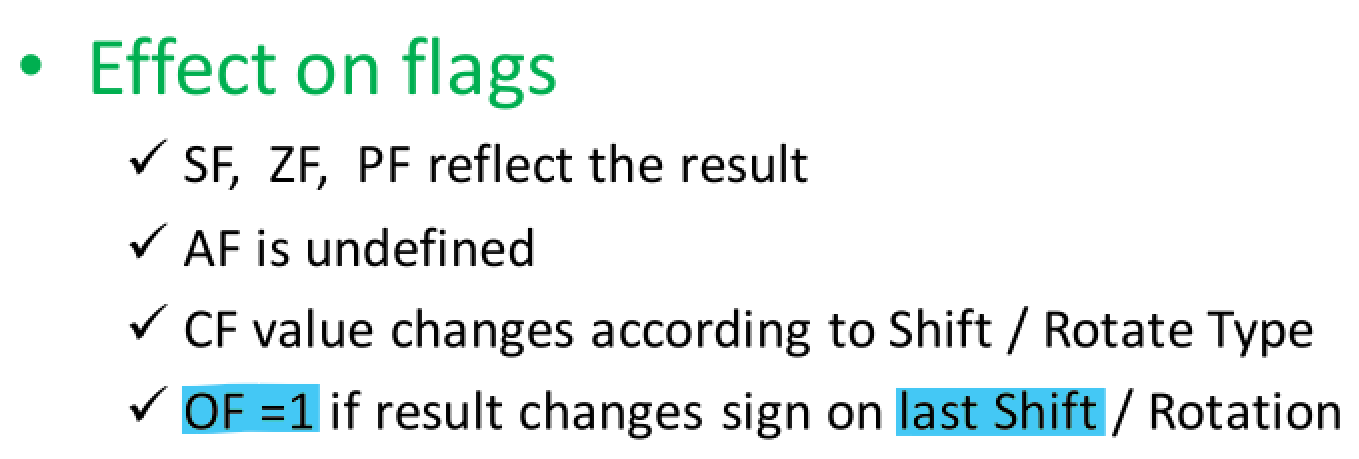
TEST

performs and operation , but doesn't write to destination

```
TEST AL , 1 ; check if odd or even
```

Shift Operation

- `OPCODE destination , 1` : does only 1 shift operation
- `OPCODE destination , CL` : does CL amount of shift operation



- Effect on flags
  - ✓ SF, ZF, PF reflect the result
  - ✓ AF is undefined
  - ✓ CF value changes according to Shift / Rotate Type
  - ✓ OF =1 if result changes sign on last Shift / Rotation

SHL & SAL

```
MOV AL , 10D
SHL AL , 1

MOV CL , 2
SHL AL , CL
```

Note

CF and OF are not reliable for multiple shifts because multiple shifts are just a series of single shifts and **CF, OF only reflect the result of the last shift**

SHR & SAR

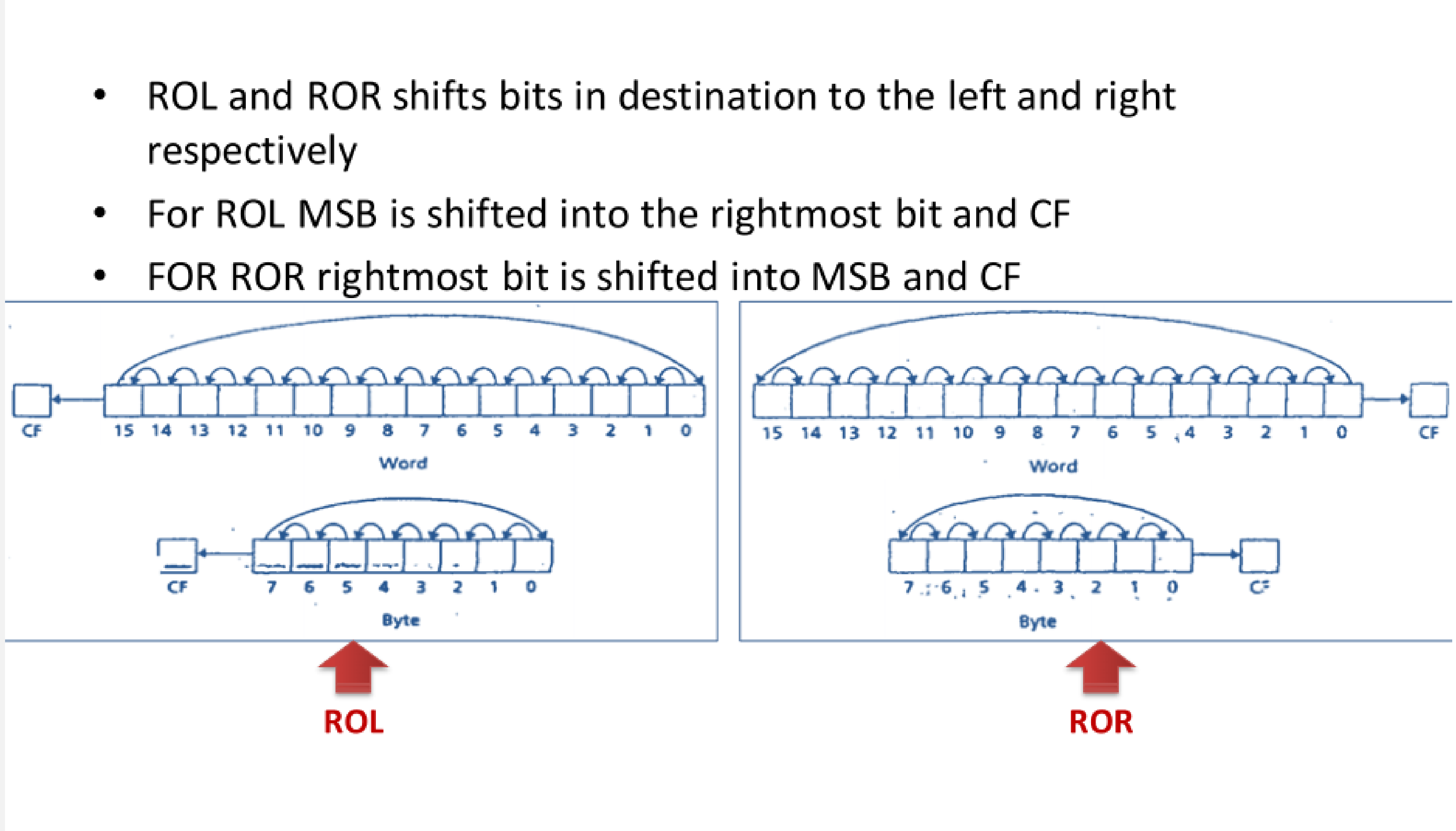
```
MOV AL , 105D
SHR AL , 1

MOV CL , 2
SHR AL , CL
```

Note

- In case of SAR MSB retains its original value
- `SHR` should be used for `unsigned interpretation` as it does not preserve sign
- `SAR` should be used for `signed interpretation` as it preserves sign

ROL & ROR

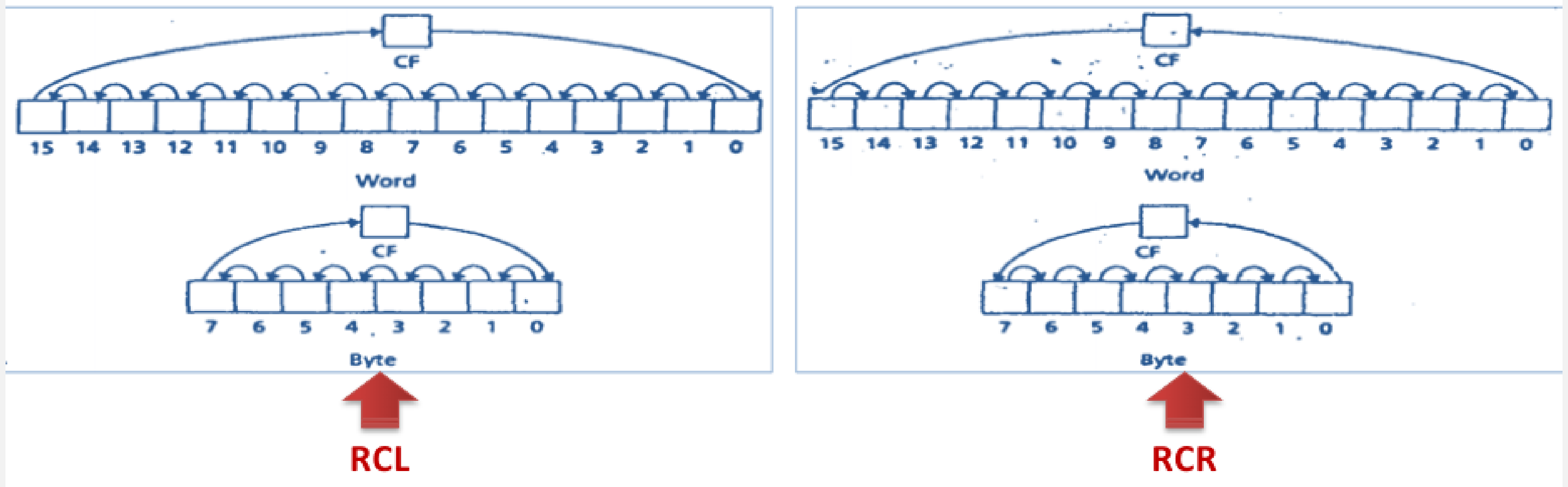
- rolling shift left / right



- ROL and ROR shifts bits in destination to the left and right respectively
- For ROL MSB is shifted into the rightmost bit and CF
- FOR ROR rightmost bit is shifted into MSB and CF

RCL & RCR

- Works similarly to ROL and ROR respectively
- For RCL, CF is shifted to LSB and MSB is shifted to CF
- FOR RCR, CF is shifted to MSB and LSB shifted to CF



MULTIPLICATION

- `MUL` : unsigned
- `IMUL` : signed

How to multiply A x B

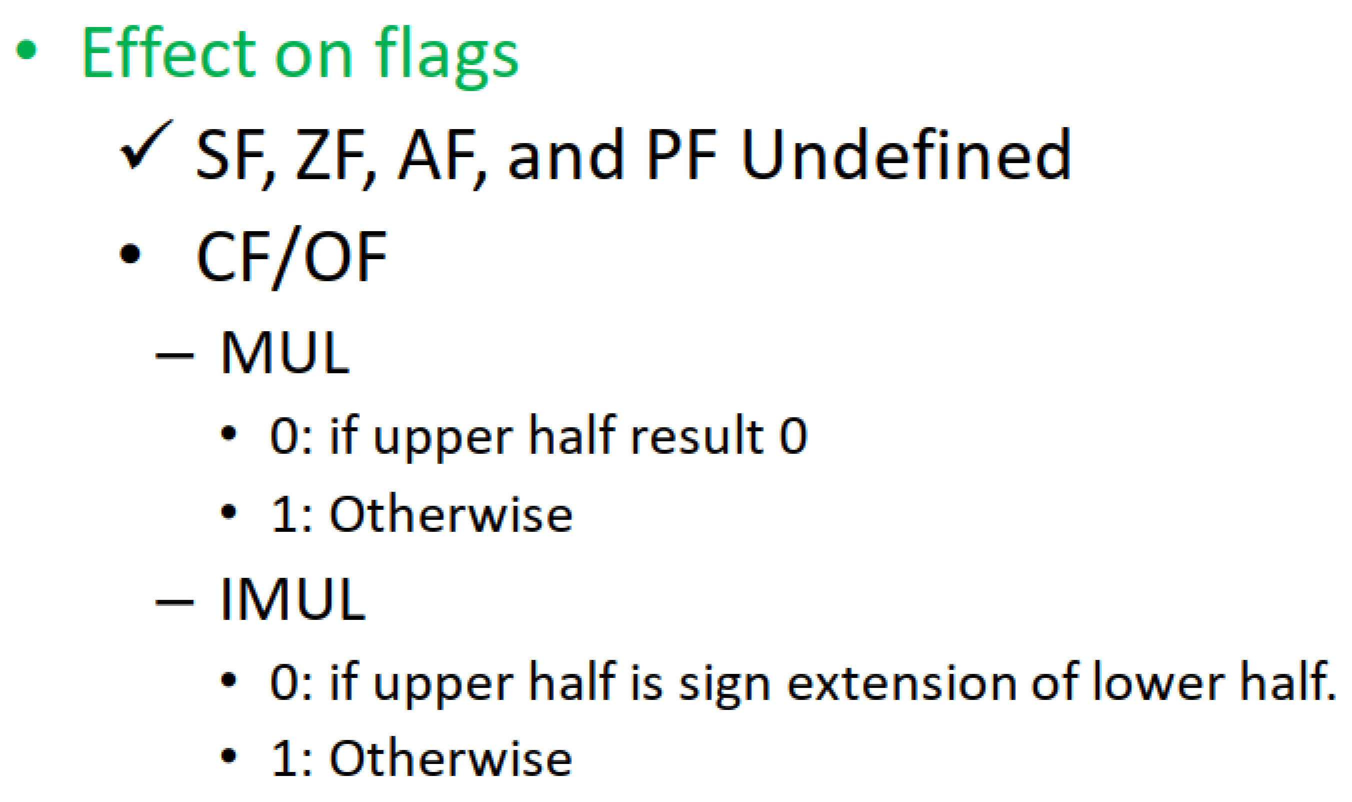
`bytes`

- store A in a 8 bit register
- store B in `AL`
- product stored in `AX`

`words`

- store A in 16 bit register
- store B in `AX`
- product stored in
  - MSB (16 bit) : `DX`
  - LSB (16 bit) : `AX`

Effect on flag



- Effect on flags
  - ✓ SF, ZF, AF, and PF Undefined
  - CF/OF
    - MUL
      - 0: if upper half result 0
      - 1: Otherwise
    - IMUL
      - 0: if upper half is sign extension of lower half.
      - 1: Otherwise

Division

- `DIV` : unsigned
- `IDIV` : signed

How to multiply B / A

`bytes`

- store A in a 8 bit register
- store B in `AX`
- quotient in `AL`
- remainder in `AH`

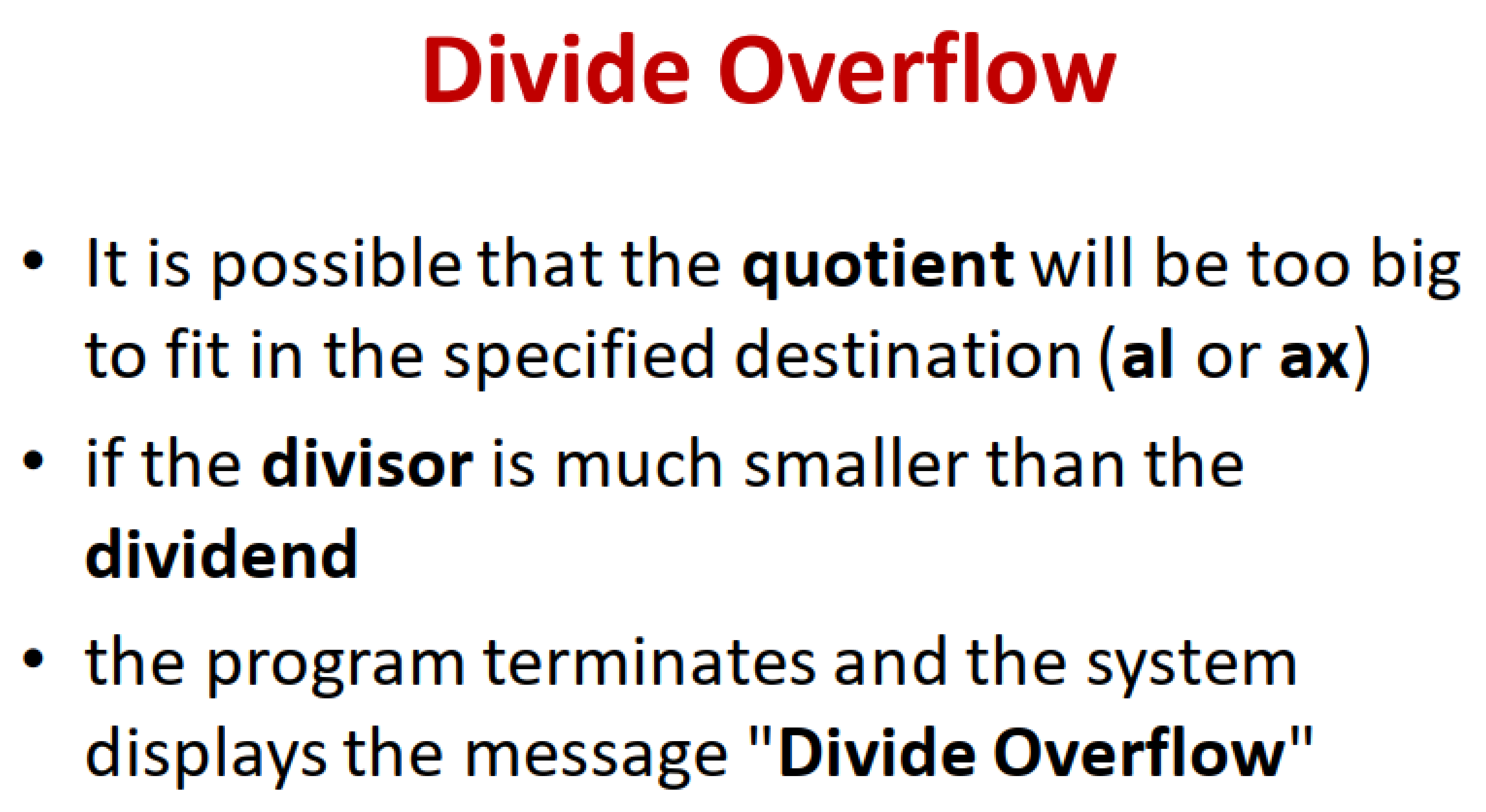
`words`

- store A in a 16 bit register
- store B in `DX:AX`
- quotient in `AX`
- remainder in `DX`

Effect on flag

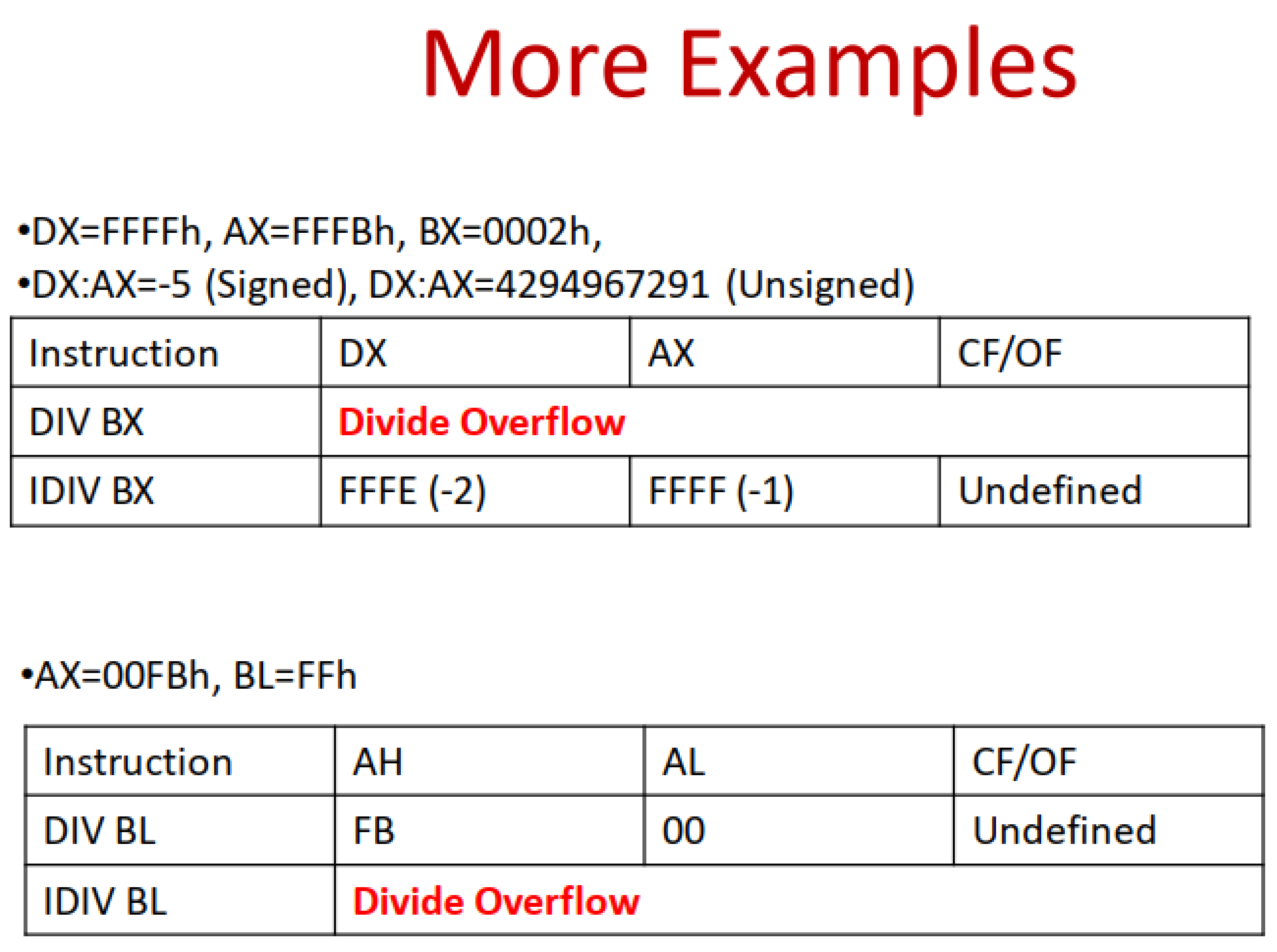
undefined

Divide Overflow



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**"



More Examples

•DX=FFFFh, AX=FFFBh, BX=0002h,
•DX:AX=-5 (Signed), DX:AX=4294967291 (Unsigned)

Instruction	DX	AX	CF/OF
DIV BX	Divide Overflow		
IDIV BX	FFFE (-2)	FFFF (-1)	Undefined

•AX=00FBh, BL=FFh

Instruction	AH	AL	CF/OF
DIV BL	FB	00	Undefined
IDIV BL	Divide Overflow		

Sign Extension (`CWD`)

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**