The Processor Status and the FLAGS Register

Course Code: COE 3205

Course Title: Computer Organization & Architecture

Dept. of Computer Science
Faculty of Science and Technology

Lecturer No:	**12**	**Week No:**	**13**	**Semester:**	
Lecturer:					

Lecture Outline

1. **Overview**
2. **Multiplication**
3. **Division**

Overview

- In this chapter, we introduce Instructions for multiplying and dividing any numbers
- The process of multiplication and division is different for signed and unsigned numbers. So there are different Instructions for signed and unsigned multiplication and division.
- Again, these instructions have byte and word forms

Signed versus Unsigned Multiplication

- MUL and IMUL
- Syntax of MUL and IMUL
- Byte Form Multiplication
- Word Form Multiplication
- Effect of MUU/MUL on the status flags
- Simple Applications· of MUL and IMUL

Signed versus Unsigned Multiplication

- In binary multiplication, signed· and unsigned numbers must be treated differently.

- For example, suppose we want to multiply the eight-bit numbers **10000000** and **11111111**. Interpreted as unsigned numbers, they represent **128** and **255** respectively.

- The product is **32,640** = **0111111110000000b**.

- However, taken as signed numbers, they represent-**128** and **-1**, respectively; and the product is **128** = **0000000010000000b**

Signed versus Unsigned Multiplication

MUL and IMUL

- Since signed and unsigned multiplication lead to different results. There are **two** multiplication Instructions:
    - MUL (multiply for unsigned multiplication.
    - IMUL for signed multiplication.

Signed versus Unsigned Multiplication

Syntax of MUL and IMUL

- **MUL source**
- **IMUL Source**

Signed versus Unsigned Multiplication

Byte and Word Form of MUL and IMUL

- MUL and IMUL instructions multiply **bytes** or **words**.

- **Byte Form:** If two **bytes** are multiplied, then product is a **word** (**16 bits**).

- **Word Form:** If two **words** are multiplied, the product is a **doubleword (32 bits**).

Signed versus Unsigned Multiplication

Byte Form Multiplication

- For **byte** multiplication, one number is contained in the source and the other is assumed to be in **AL**.
- The **16-bit** product will be in **AX**.
- The source may be a **byte register** or **memory** byte, but cannot be a **constant**

Signed versus Unsigned Multiplication

Byte Form Multiplication

- For **word form** multiplication, one number is contained in the source and the other is assumed to be in **AX**.

- The most significant **16 bits** of the **doubleword** product will be in DX,
- And the least significant **16 bits** will be in **AX**

- Sometimes it is written as **DX:AX**.

- The source may be a **16-bit register** or **memory** word, but not a· **constant**.

Signed versus Unsigned Multiplication

Effect of MUU/MUL on the status flags

- **SF,ZF,AF, PF: undefined**

- **CF/OF:**
  - After **MUL**, CF/OF : = **0** if the upper half of the result is· zero , otherwise CF/OF = **1**

  - After **IMUL**, CF/OF = **0** if the upper half of the result is the sign extension of the lower half (this means that the bits of the upper half are the same as the sign bit of the lower half). = otherwise CF/OF = **1**.

Signed versus Unsigned Multiplication

Meaning of CF/OF = 1 on MUL and IMUL

- For both **MUL** and **IMUL**, **CF/OF = l** means that the product is too big to fit' in 'the lower half of the **destination** (**AL** for **byte** multiplication, **AX** for **word** multiplication).

Signed versus Unsigned Multiplication

Example of Multiplication

Example 1: Suppose AX contains, **1** and BX contains **FFFFh**

Instructions	Decimal Product	Hex Product	DX	AX	CF/OF
MUL BX	65535	0000FFFF	0000	FFFF	0
IMUL BX	-1	FFFFFFFF	FFFF	FFFF	0

- For **MUL**. **DX = 0,** so CF/OF= **0**.
- for **IMUL**, the signed interpretation of BX is **-1**, and the product is also **-1**. In 32 bits, this is FFFFFFFFh. CF/OF = 0 because DX is the sign extension of AX.

Signed versus Unsigned Multiplication

Example of Multiplication

Example 9.1: Suppose AX contains, **1** and BX contains **FFFFh**

Instructions	**Decimal Product**	**Hex Product**	**DX**	**AX**	**CF/OF**
MUL BX	65535	0000FFFF	0000	FFFF	0
IMUL BX	-1	FFFFFFFF	FFFF	FFFF	0

- For **MUL**. **DX = 0,** so CF/OF= **0**.
- for **IMUL**, the signed interpretation of BX is **-1**, and the product is also **-1**. In 32 bits, this is FFFFFFFFh. CF/OF = 0 because DX is the sign extension of AX.

Signed versus Unsigned Multiplication

Example of Multiplication

Example 9.2 Suppose AX contains FFFFh and BX contains FFFFh:

Instruction	*Decimal product*	*Hex product*	*DX*	*AX*	*CF/OF*
MUL BX	4294836225	FFFE0001	FFFE	0001	1
IMUL BX	1	00000001	0000	0001	0

For MUL, CF/OF = 1 because DX is not 0. This reflects the fact that the product FFFE0001h is too big to fit in AX.

For IMUL, AX and BX both contain –1, so the product is 1. DX has the sign extension of AX, so CF/OF = 0.

Signed versus Unsigned Multiplication

Example of Multiplication

Example 9.3 Suppose AX contains 0FFFh:

Instruction	*Decimal product*	*Hex product*	*DX*	*AX*	*CF/OF*
MUL AX	16769025	00FFE001	00FF	E001	1
IMUL AX	16769025	00FFE001	00FF	E001	1

Because the msb of AX is 0, both MUL and IMUL give the same product. Because the product is too big to fit in AX, CF/OF = 1.

Signed versus Unsigned Multiplication

Example of Multiplication

Example 9.4 Suppose AX contains 0100h and CX contains FFFFh:

Instruction	*Decimal product*	*Hex product*	*DX*	*AX*	*CF/OF*
MUL CX	16776960	00FFFF00	00FF	FF00	1
IMUL CX	-256	FFFFFF00	FFFF	FF00	0

For MUL, the product FFFF00 is obtained by attaching two zeros to the source value FFFFh. Because the product is too big to fit in AX, CF/OF = 1.

For IMUL, AX contains 256 and CX contains -1, so the product is -256, which may be expressed as FF00h in 16 bits. DX has the sign extension of AX, so CF/OF = 0.

Signed versus Unsigned Multiplication

Example of Multiplication

Example 9.5 Suppose AL contains 80h and BL contains FFh:

Instruction	*Decimal product*	*Hex product*	*AH*	*AL*	*CF/OF*
MUL BL	128	7F80	7F	80	1
IMUL BL	128	0080	00	80	1

For byte multiplication, the 16-bit product is contained in AX.

For MUL, the product is 7F80. Because the high eight bits are not 0, CF/OF = 1.

For IMUL, we have a curious situation. 80h = −128, FFh = −1, so the product is 128 = 0080h. AH does not have the sign extension of AL, so CF/OF = 1. This reflects the fact that AL does not contain the correct answer in a signed sense, because the signed decimal interpretation of 80h is −128.

Signed versus Unsigned Multiplication

Simple Applications· of MUL and IMUL

Example 9.5 Suppose AL contains 80h and BL contains FFh:

Instruction	*Decimal product*	*Hex product*	*AH*	*AL*	*CF/OF*
MUL BL	128	7F80	7F	80	1
IMUL BL	128	0080	00	80	1

For byte multiplication, the 16-bit product is contained in AX.

For MUL, the product is 7F80. Because the high eight bits are not 0, CF/OF = 1.

For IMUL, we have a curious situation. 80h = −128, FFh = −1, so the product is 128 = 0080h. AH does not have the sign extension of AL, so CF/OF = 1. This reflects the fact that AL does not contain the correct answer in a signed sense, because the signed decimal interpretation of 80h is −128.

Signed versus Unsigned Multiplication

Simple Applications (1) of MUL and IMUL

Example 9.6 Translate the high-level language assignment statement A = 5 × A − 12 × B into assembly code. Let A and B be word variables, and suppose there is no overflow. Use IMUL for multiplication.

Solution:

```
MOV   AX,5          ;AX = 5
IMUL  A             ;AX = 5 x A
MOV   A,AX          ;A = 5 x A
MOV   AX,12         ;AX = 12
IMUL  B             ;AX = 12 x B
SUB   A,AX          ;A = 5 x A - 12 x B
```

Signed versus Unsigned Multiplication

Simple Applications (2) of MUL and IMUL continues…

Example 9.7 Write a procedure FACTORIAL that will compute $N!$ for a positive integer N. The procedure should receive N in CX and return $N!$ in AX. Suppose that overflow does not occur.

Solution: The definition of $N!$ is

$$N! = 1 \text{ if } N = 1$$
$$= N \times (N - 1) \times (N - 2) \times .. \times 1 \text{ if } N > 1$$

Here is an algorithm: -

```
product = 1
term = N
FOR N times DO
 product = product x term
 term = term - 1
ENDFOR
```

Signed versus Unsigned Multiplication

Simple Applications of MUL and IMUL

It can be coded as follows:

```
FACTORIAL       PROC
;computes N!
;input: CX = N
;output: AX = N!
                MOV   AX,1       ;AX holds product
TOP:
                MUL   CX         ;product = product x term
                LOOP  TOP
                RET
FACTORIAL       ENDP
```

Here CX is both loop counter and term; the LOOP instruction automatically decrements it on each iteration through the loop. We assume the product does not overflow 16 bits.

Signed versus Unsigned Division

Division Syntax

- Division Syntax
- Division Byte Form
- Division Word Form
- Division Overflow
- Sign Extension of the Dividend

Signed versus Unsigned Division

Division Syntax

- When division is performed, we obtain two results, the **quotient** and the **remainder**.

- As with multiplication, there are separate Instructions for **unsigned** and **signed** division

- **DIV** (divide) is used for **unsigned** division

- **IDIV-**(lnteger divide) is used for **signed** division.

- **The syntax :**
    - DIV divisor
    - IDIV divisor

Signed versus Unsigned Division

Division Byte Form

- The **divisor** is an **8-bit** register or memory byte.
- The **16-bit dividend** is assumed to be in **AX**.
- After division, the **8-bit quotient** is in **AL** and the **8-bit remainder is** in **AH**.
- The **divisor** may not be a **constant**

Signed versus Unsigned Division

Division Word Form

- Here the divisor is a **16-bit** register or memory word
- The 32-bit dividend is assumed to be in **DX:AX**
- After division, the 16-bit quotient is nn **AX** and the 16-bit **remainder** is in **DX**.
- The divisor may not he a constant.
- For **signed** division, the remainder has the same sign as the dividend. If both dividend and divisor are positive, **DIV** and **IDIV** give the same result.
- The effect of **DIV /IDIV** on the **flags** is that all status flags are **undefined**

Signed versus Unsigned Division

Division Overflow

- It is possible that the **quotient** will be too big to fit in the specified destination· (**AL** or **AX**).
- This can happen if the **divisor** is much smaller than the **dividend**
- When this happens, the program terminates and the system displays the message "**Divide Overflow**"

Signed versus Unsigned Division

Division Example

Example 9.8 Suppose DX contains 0000h, AX contains 0005h, and BX contains 0002h.

Instruction	*Decimal quotient*	*Decimal remainder*	*AX*	*DX*
DIV BX	2	1	0002	0001
IDIV BX	2	1	0002	0001

Dividing 5 by 2 yields a quotient of 2 and a remainder of 1. Because both dividend and divisor are positive, DIV and IDIV give the same results.

Signed versus Unsigned Division

Division Example

Example 9.9 Suppose DX contains 0000h, AX contains 0005h, and BX contains FFFEh.

Instruction	*Decimal quotient*	*Decimal remainder*	*AX*	*DX*
DIV BX	0	5	0000	0005
IDIV BX	–2	1	FFFE	0001

For DIV, the dividend is 5 and the divisor is FFFEh = 65534; 5 divided by 65534 yields a quotient of 0 and a remainder of 5.

For IDIV, the dividend is 5 and the divisor is FFFEh = –2; 5 divided by –2 gives a quotient of –2 and a remainder of 1.

Signed versus Unsigned Division

Division Example

Example 9.9 Suppose DX contains 0000h, AX contains 0005h, and BX contains FFFEh.

Instruction	*Decimal quotient*	*Decimal remainder*	*AX*	*DX*
DIV BX	0	5	0000	0005
IDIV BX	-2	1	FFFE	0001

For DIV, the dividend is 5 and the divisor is FFFEh = 65534; 5 divided by 65534 yields a quotient of 0 and a remainder of 5.

For IDIV, the dividend is 5 and the divisor is FFFEh = -2; 5 divided by -2 gives a quotient of -2 and a remainder of 1.

Signed versus Unsigned Division

Division Example

Example 9.10 Suppose DX contains FFFFh, AX contains FFFBh, and BX contains 0002.

Instruction	*Decimal quotient*	*Decimal remainder*	*AX*	*DX*
IDIV BX	−2	−1	FFFE	FFFF
DIV BX	DIVIDE OVERFLOW			

For IDIV, DX:AX = FFFFFFFBh = −5, BX = 2. −5 divided by 2 gives a quotient of −2 = FFFEh and a remainder of −1 = FFFFh.

For DIV, the dividend DX:AX = FFFFFFFBh = 4294967291 and the divisor = 2. The actual quotient is 2147483646 = 7FFFFFFEh. This is too big to fit in AX, so the computer prints DIVIDE OVERFLOW and the program terminates. This shows what can happen if the divisor is a lot smaller than the dividend.

Signed versus Unsigned Division

Division Example

Example 9.11 Suppose AX contains 00FBh and BL contains FFh.

Instruction	*Decimal quotient*	*Decimal remainder*	*AX*	*AL*
`DIV BL`	0	251	FB	00
`IDIV BL`	DIVIDE OVERFLOW			

For byte division, the dividend is in AX; the quotient is in AL and the remainder in AH.

For DIV, the dividend is 00FBh = 251 and the divisor is FFh = 256. Dividing 251 by 256 yields a quotient of 0 and a remainder of 251 = FBh.

For IDIV, the dividend is 00FBh = 251 and the divisor is FFh = −1. Dividing 251 by −1 yields a quotient of −251, which is too big to fit in AL, so the message DIVIDE OVERFLOW is printed.

Signed versus Unsigned Division

Sign Extension of the Dividend

Word Division

- In **word** division, the dividend Is in **DX:AX** even if the actual dividend will fit in AX. In this case **DX** should be prepared as follows:

  - 1. For **DIV**, **DX** should be cleared.

  - 2. For **IDIV, DX** should be made the **sign extension** of **AX**.

- The instruction **CWD** (convert word to doubleword) will do the extension.

Signed versus Unsigned Division

Sign Extension of the Dividend - Example

Example 9.12 Divide –1250 by 7:

Solution:

```
MOV   AX,-1250          ;AX gets dividend
CWD                     ;Extend sign to DX
MOV   BX,7              ;BX has divisor
IDIV BX                 ;AX gets quotient, DX has remainder
```

Signed versus Unsigned Division

Sign Extension of the Dividend – Byte Division

Byte Division

- In the **byte** division, dividend is in **AX**.
- If the actual dividend is a byte; then **AH** should be prepared as follows:
  1. For **DIV**, **AH** should be cleared.
  2. For **IDIV AH**, should the sign extension of **AL**.

- The instruction **CBW** (convert byte to word) will do the extension.

Signed versus Unsigned Division

Sign Extension of the Dividend – Byte Division example

Example 9.13 Divide the signed value of the byte variable XBYTE by −7

Solution:

```
MOV   AL,XBYTE        ;AL has dividend
CBW                   ;Extend sign to AH
MOV   BL,-7           ;BL has divisor
IDIV  BL              ;AL has quotient, AH has remainder
```

There is no effect of CBW and CWD on the flags.

Books

- Assembly Language Programing and Organization of the IBM PC

Ytha Yu
Charles Marut

References

- Multiplication and Division Instrution in Assembly
    - https://www.youtube.com/watch?v=LDXI8OW7kfk&t=348s
- Carry and Overflow Details
    - https://www.youtube.com/watch?v=9cXe_T99nL4