



Microprocessors and Interfaces:

Lecture 13

8086 Arithmetic Instructions : Part-2

By Dr. Sanjay Vidhyadharan

Reference : UNDERGRADUATE TEXTS IN COMPUTER SCIENCE

Springer Science+Business Media, LLC

Editors

David Gries

Fred B. Schneider

Arithmetic Instructions

- Addition
- Subtraction
- Increment
- Decrement
- Comparison
- Multiplication
- Division
- Decimal Adjust

saralavidhyadharan.in

Multiplication

- Unsigned Multiplication (mul)

$$\begin{array}{rcl} 11111111 & \times & 11111111 \\ (255D) & & (255D) \\ \hline & & 11111110\ 11111111 \\ & & (65025D) \end{array}$$

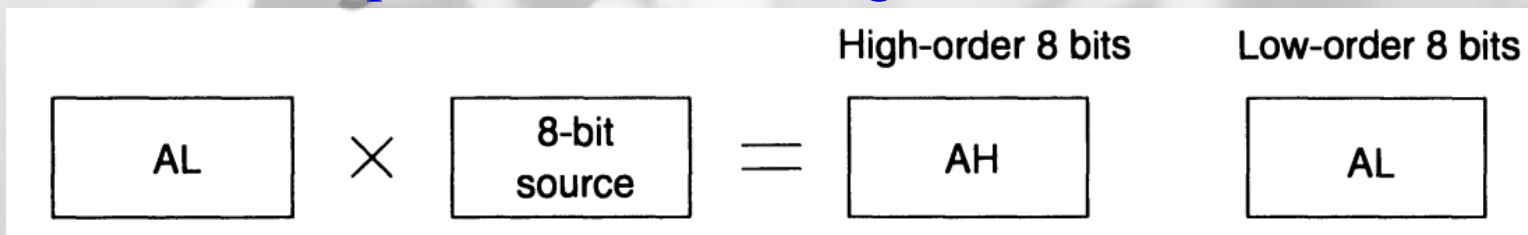
- Signed Multiplication (imul)

$$11111111 \times 11111111 = 00000000\ 00000001$$

8 Bit X 8 bit – Product will be 16-bit

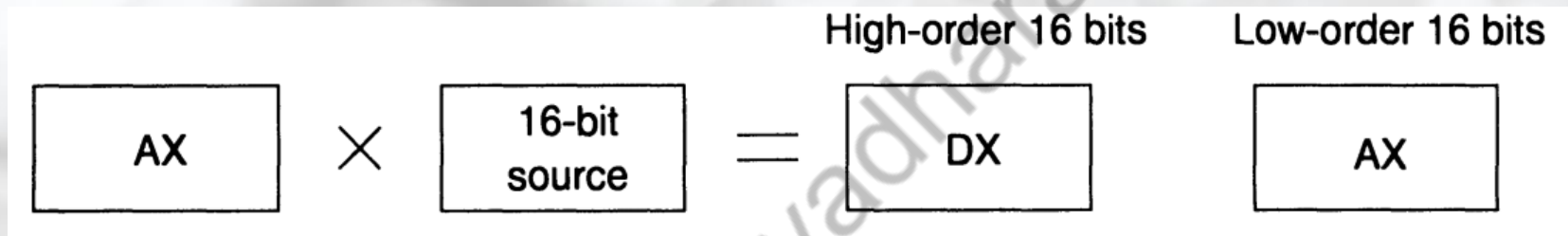
- MUL Source (Product in AX)
- **MUL 10** (invalid)
- MUL Byte PTR [BX] (Product in AX)

The 16-bit result is placed in the AX register

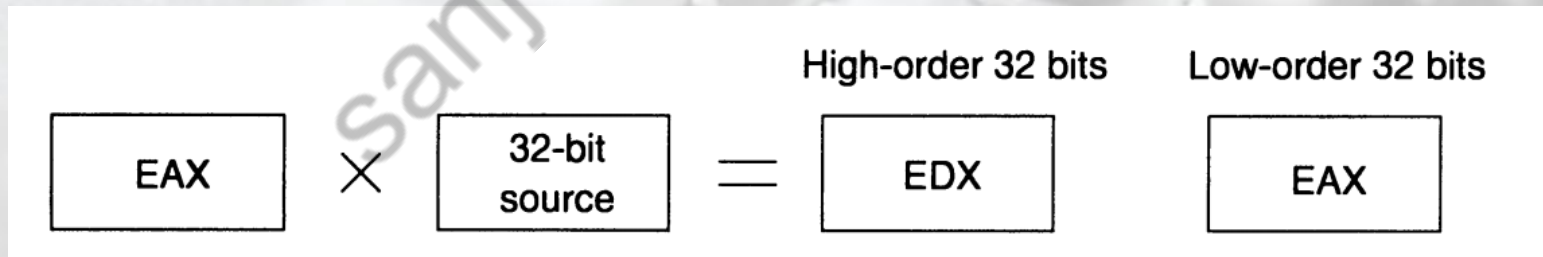


Multiplication

If the source operand is a word, it is multiplied by the contents of the AX register and the doubleword result is placed in the DX and AX register pair, with the AX register holding the lower-order 16 bits, as shown below

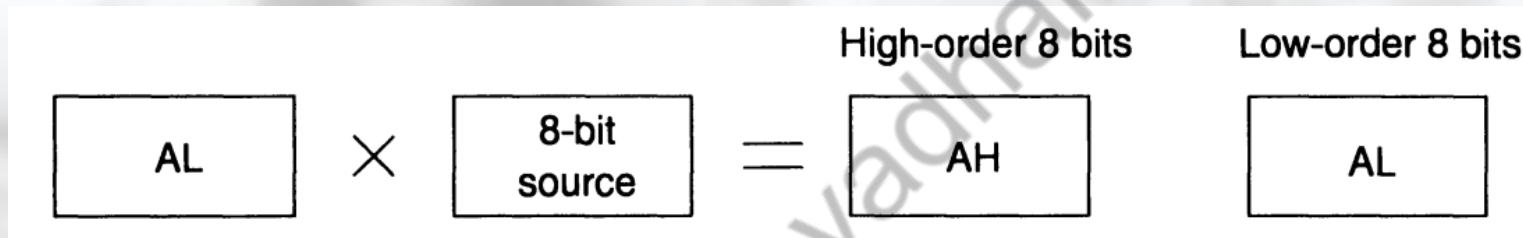


If the source operand is a double word, it is multiplied by the contents of the EAX register and the 64-bit result is placed in the EDX and EAX register pair, with the EAX register holding the lower-order 32 bits, as shown below.



Multiplication

Flags : The mul instruction affects all six status flags. However, it updates only the carry and overflow flags. **The remaining four flags are undefined.** Setting of the carry and overflow flags does not indicate an error condition. Instead, this condition implies that AH, DX, or EDX contains significant digits of the result.



Example 1:

```
mov AL, 10  
mov DL, 25  
mul DL
```

Carry and overflow not set (8-bit result, AH will have 00000000)

Example 2:

```
mov AL, 10  
mov DL, 26  
mul DL
```

Carry and overflow set (16-bit result)

Multiplication

Example 3

Multiply two 8-bit numbers, where numbers are stored from offset 100 and store the result into offset 200.

```
MOV SI, 100  
MOV DI, 200  
MOV AL, [SI]  
INC SI  
MUL BYTE PTR [SI]  
MOV [DI], AX
```

Multiplication

Signed Multiplication :

imul source

The carry and overflow flags are set if the upper half of the result is not the sign extension of the lower half

Example 1:

```
mov DL, 0FFH;    DL = -1
mov AL, 42H;     AL = 66
imul DL ;        AH-AL: 1111111-10111110
```

Carry and overflow will be reset (8-bit result, AH is sign extended)

Example 2:

```
mov DL, 0FFH;    DL = -1
mov AL, 0BEH;    AL = -66
imul DL ;        AH-AL: 00000000-01000010
```

Carry and overflow will be reset (8-bit result, AH is sign extended)

Example 3:

```
mov DL, 25H;     DL = 25
mov AL, 0F6H;    AL = -10
imul DL ;        AH-AL: 11111111-00000110
```

Carry and overflow will be Set (16-bit result, AH is **not** sign extended)

Division

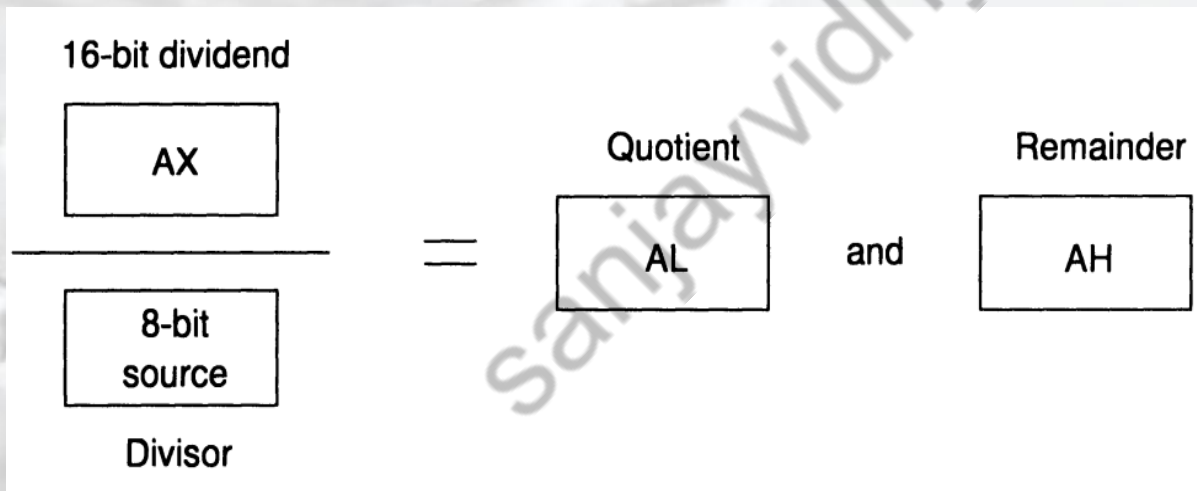
div source (unsigned)

idiv source (signed)

Division generates two result components-quotient and remainder.

In multiplication, by using double-length registers, no overflow occurs. In division, divide overflow is a real possibility and Pentium generates a special software interrupt when a divide overflow occurs.

If the source operand is a byte, the dividend is assumed to be in the AX register and 16 bits long. After division, the quotient is returned in the AL register and the remainder in the AH register, as shown below.



Example 251/12

```
mov AX, 00FBH (251 D)
```

```
mov CL, 0CH (12D)
```

```
div CL
```

AL := 14H (20D) Quotient

AH := 0BH (11D) Remainder

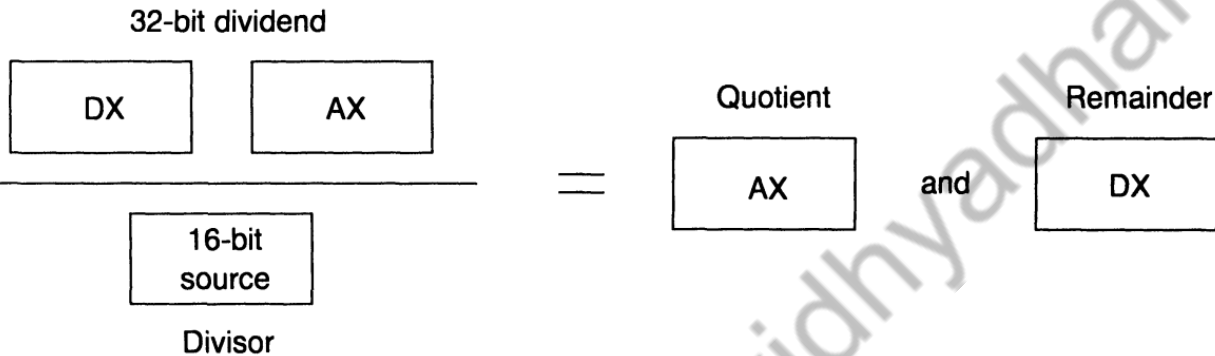
All six status flags are affected and are undefined.

Division

div source (unsigned)

idiv source (signed)

If the source operand is a word, the dividend is assumed 32 bits long and in the DX and AX registers (upper 16 bits in DX). After the division, the 16-bit quotient will be in AX and the 16-bit remainder in DX, as shown below.

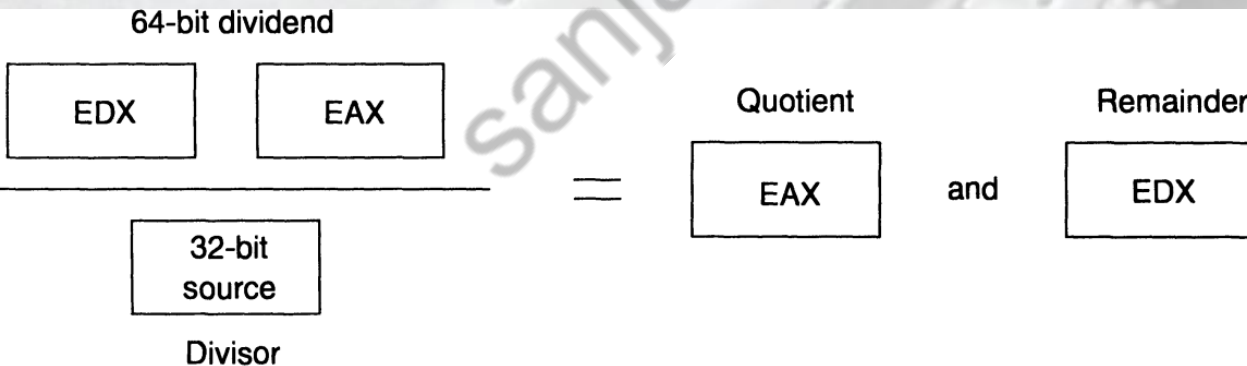


Example 5147/300

```
mov DX, 0
mov AX, 141BH (5147D)
mov CX, 012CH (300D)
div CX
```

AX := 0012H (17D) Quotient
DX := 002FH (47D) Remainder

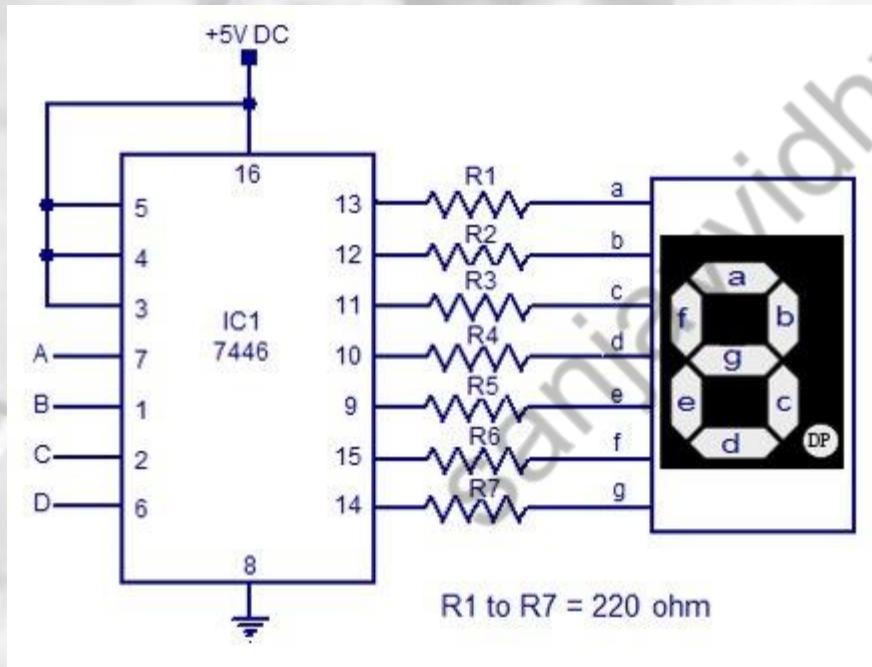
If source operand is a double word



Binary Coded Decimal

General digital systems

User enters decimal \rightarrow BCD i/p \rightarrow Binary i/p \rightarrow compute in binary \rightarrow Binary o/p \rightarrow BCD o/p \rightarrow Decimal output shown to user



Binary Coded Decimal

BCD addition

4 + 5

4 0 1 0 0

5 0 1 0 1

9 1 0 0 1

Expected Result

4 + 8

4 0 1 0 0

8 1 0 0 0

1 1 0 0

Is this expected Result ?

Expected answer
in BCD : 12

0001 : 0010

Binary Coded Decimal

BCD addition

9 + 9

9 1 0 0 1

9 1 0 0 1

Carry out generated

1 0 0 1 0

Expected result ?

0 1 1 0

Add correction of +6

0 0 0 1 - 1 0 0 0

1

8

DAA

DAA follows the ADD or ADC instruction to adjust the result into a BCD result.

Example

```
MOV AL, 01H
```

```
ADD AL, 09H
```

```
DAA
```

AX=0 AL=A (1010)

AX=1 AL=0 (1010+0110)

AF, CF, PF and ZF are affected. OF is undefined after DAA instruction.

DAS follows the SUB or SBB instruction to adjust the result into a BCD result.

ASCII Arithmetic

ASCII Code: Character to Binary

0	0011 0000	O	0100 1111	m	0110 1101
1	0011 0001	P	0101 0000	n	0110 1110
2	0011 0010	Q	0101 0001	o	0110 1111
3	0011 0011	R	0101 0010	p	0111 0000
4	0011 0100	S	0101 0011	q	0111 0001
5	0011 0101	T	0101 0100	r	0111 0010
6	0011 0110	U	0101 0101	s	0111 0011
7	0011 0111	V	0101 0110	t	0111 0100
8	0011 1000	W	0101 0111	u	0111 0101
9	0011 1001	X	0101 1000	v	0111 0110
A	0100 0001	Y	0101 1001	w	0111 0111
B	0100 0010	Z	0101 1010	x	0111 1000
C	0100 0011	a	0110 0001	y	0111 1001
D	0100 0100	b	0110 0010	z	0111 1010
E	0100 0101	c	0110 0011	.	0010 1110
F	0100 0110	d	0110 0100	,	0010 0111
G	0100 0111	e	0110 0101	:	0011 1010
H	0100 1000	f	0110 0110	;	0011 1011
I	0100 1001	g	0110 0111	?	0011 1111
J	0100 1010	h	0110 1000	!	0010 0001
K	0100 1011	I	0110 1001	'	0010 1100
L	0100 1100	j	0110 1010	"	0010 0010
M	0100 1101	k	0110 1011	(0010 1000
N	0100 1110	l	0110 1100)	0010 1001
				space	0010 0000

ASCII Arithmetic

- Four instructions in ASCII arithmetic operations:
 - AAA (ASCII adjust after addition)
 - AAD (ASCII adjust before division)
 - AAM (ASCII adjust after multiplication)
 - AAS (ASCII adjust after subtraction)

AAA Instruction

- Addition of two one-digit ASCII-coded numbers will not result in any useful data.
- Ex: Before: AL= 0011 0001 , ASCII 1;
BL= 0011 1001, ASCII 9
ADD AL,BL ; Result : AL=0110 1110 = 6AH,
; which is incorrect ASCII
AAA ; AH 0001 AL 0001
ADD AX, 3030
- ✓ The AAA instruction works only **on the AL register**.
- ✓ The AAA instruction updates AF and CF but OF,PF,SF and ZF are left undefined.
- **AAS adjusts the AX register after an ASCII subtraction.**

AAM (BCD Adjust after multiply)

- Follows multiplication instruction after multiplying two one-digit unpacked BCD numbers.
- AAM converts from binary to unpacked BCD.
- Ex: AL= 00000101 =unpacked BCD 5

BH=00001001 = unpacked BCD 9

MUL BH ; AL X BH, result in AX

; AX =00000000 00101101 =002DH

AAM ; AX=0000 0100 00000101= 0405H

; which is unpacked BCD for 45.

AAD (BCD to Binary convert before Division)

- Appears before a division.
- The AAD instruction requires the AX register contain a two-digit unpacked BCD number (not ASCII) before executing.
- Ex: AX= 0607H unpacked BCD for 67 decimal

CH=09 H , now adjust to binary

AAD ; result: AX=0043=43H= 67 decimal

DIV CH ; Divide AX by unpacked BCD in CH

; quotient : AL=07 unpacked BCD

; Remainder : AH=04 unpacked BCD

; Flags undefined after DIV

AAS Instruction

- AAS adjusts the AX register after an ASCII subtraction.
- Ex1: AL=00111001 =39H =ASCII 9

BL= 00110101 =35H= ASCII 5

SUB AL,BL ;Result: AL= 00000100= BCD 04
and CF=0

AAS ; result: AL=00000100 = BCD 04
and CF=0, no borrow required.
ASCII 5 - ASCII 9(5-9)



Thankyou