

386 Instruction Set

- 9 Operation Categories
 - Data Transfer
 - Arithmetic
 - Shift/Rotate
 - String Manipulation
 - Bit Manipulation
 - Control Transfer
 - High Level Language Support
 - Operating System Support
 - Processor Control
- Number of operands:
0, 1, 2, or 3

TABLE 2-20. ADDITIONAL INSTRUCTIONS

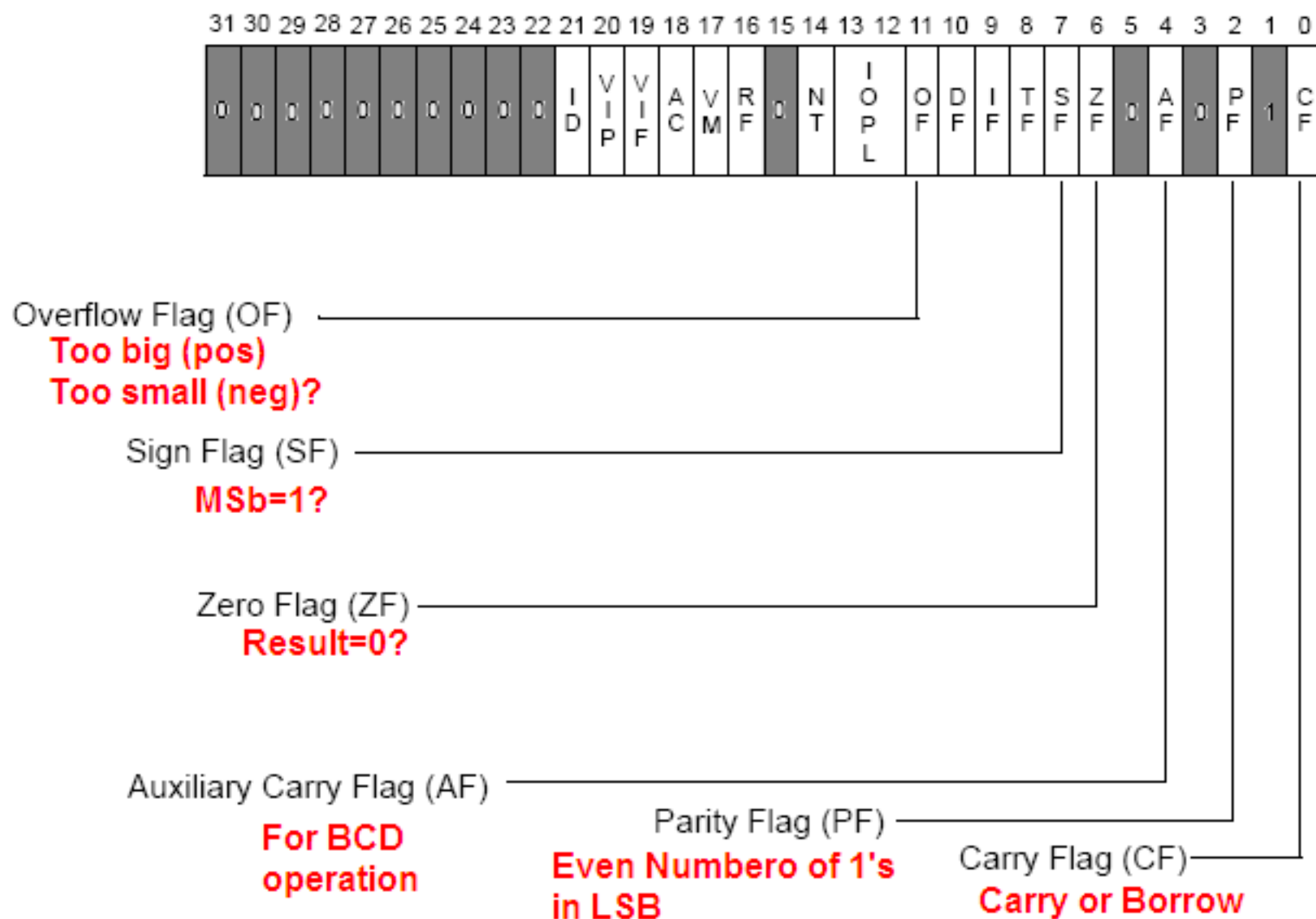
ADDITION	
ADD	Add operands
ADC	Add with carry
INC	Increment operand by 1
AAA	ASCII adjust for addition
DAA	Decimal adjust for addition
SUBTRACTION	
SUB	Subtract operands
SBB	Subtract with borrow
DEC	Decrement operand by 1
NEG	Negate operand
CMP	Compare operands
DAS	Decimal adjust for subtraction
AAS	ASCII Adjust for subtraction
MULTIPLICATION	
MUL	Multiply Double/Single Precision
IMUL	Integer multiply
AAM	ASCII adjust after multiply
DIVISION	
DIV	Divide unsigned
IDIV	Integer Divide
AAD	ASCII adjust before division

Addition Instruction

- ADD (Add Integers)
 - (DST + SRC) → DST
 - replaces the destination operand with the sum of the source and destination operands. OF, SF, ZF, CF are all affected.

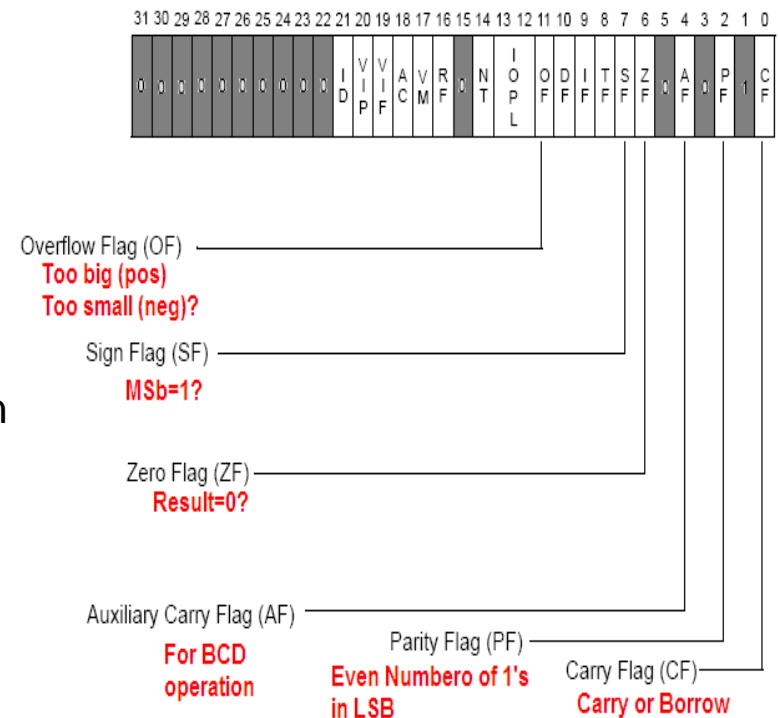
<i>Before</i>	<i>Instruction Executed</i>	<i>After</i>										
EAX: 00 00 00 75 ECX: 00 00 01 A2	add eax, ecx	<table><tr><td>EAX</td><td>00</td><td>00</td><td>02</td><td>17</td></tr><tr><td>ECX</td><td>00</td><td>00</td><td>01</td><td>A2</td></tr></table>	EAX	00	00	02	17	ECX	00	00	01	A2
EAX	00	00	02	17								
ECX	00	00	01	A2								
		SF 0 ZF 0 CF 0 OF 0										

Status Flags



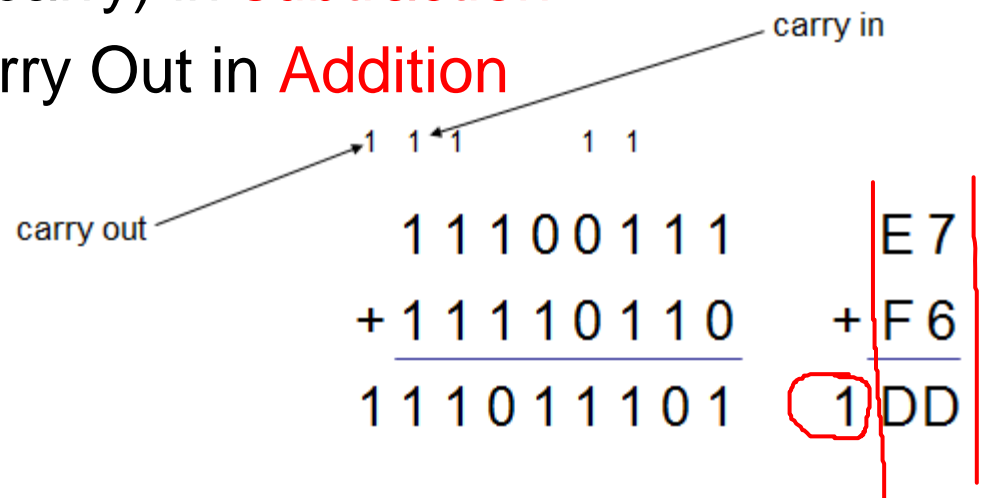
Status Flags

- CF (Carry Flag) EFL[0]
 - 1: Result of **unsigned** operation is too large
 - 0: otherwise
- PF (Parity Flag) EFL[2]
 - 1: LSB contains an even number of 1's
 - 0: odd number of 1's
- AF (Auxiliary Carry Flag) EFL[4]
 - 1: Carry from bit 3 to bit 4 in an 8-bit operation
 - 0: Otherwise
- ZF (Zero Flag) EFL[6]
 - 1: Result is zero (0)
 - 0: Non-zero
- SF (Sign Flag) EFL[7]
 - 1: Result is Negative
 - 0: Positive
- OF (Overflow Flag) EFL[11]
 - 1: Result of **signed** operation is too large
 - 0: Otherwise



Flags: CF, ZF, SF

- **SF (Sign Flag):** 1 (neg) 0 (pos)
- **ZF (Zero Flag):** 1 (result is zero) 0 (otherwise)
- **CF (Carry Flag)**
 - If the **sum** of two numbers is one bit longer than the operands, the extra 1 is a carry (or **carry out**) → CF=1
 - A 1 carried into the **highest-order (sign, leftmost) bit position** during addition is called a “**carry in**”.
 - CF=1 for borrow (or no carry) in **subtraction**.
 - CF =1 when there is Carry Out in **Addition**



Flags: CF, ZF, SF

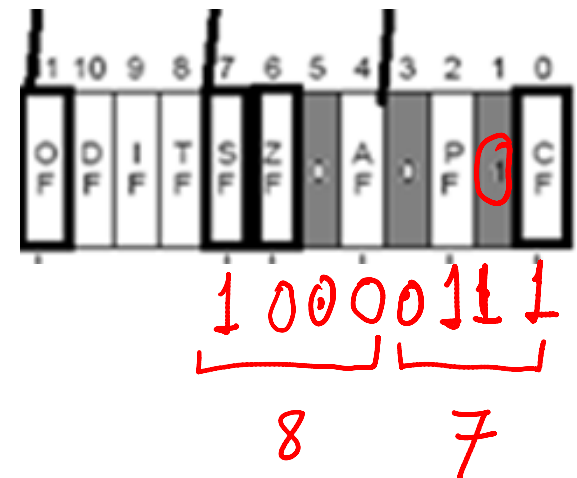
Carry Flag Example

```

mov    eax, 0
mov    ebx, 0
mov    AL, 0E7h
mov    BL, 0F6h
add    BL, AL
    
```

1 1 0 1 1 0 1
 C Sign (Neg) # of 1's 6 → Even P=1

AX
 AH AL
 BX
 BH BL
 E7 ← AL
 F6 ← BL
 1:DD (13)
 $\frac{14}{15} = 16 + \frac{13}{15}$
 1 ← carry



Flags1.asm

LAHF	Load Flags into AH Register
Transfers bits 0 to 7 of the flags register to AH.	

```
.code
main PROC
```

```

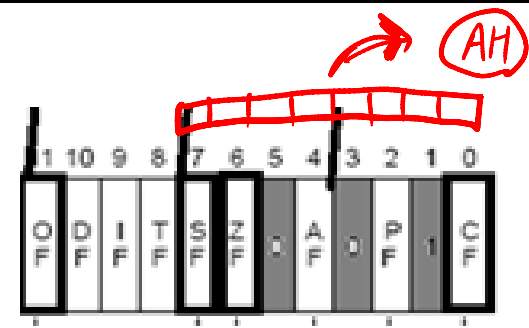
; (1)
mov EAX,0
mov EBX,0
mov AL, 0E7h
mov BL, 0F6h
add BL,AL           ; BL = E7 + F6
lahf           ; Load the Flags bits 7 - 0 to EAX
call DumpRegs   ; Show the contents of Reg and EAX

; (2)
mov EAX,0
mov EBX,0
mov AX, 483Fh

```

```
EAX=000087E7  EBX=000000DD  ECX=00000000  EDI=00401005  
ESI=00000000  EDI=00000000  EBP=0018FF94  ESP=0018FF8C  
EIP=00401026  EFL=00000287  CF=1  SF=1  ZF=0  OF=0  AF=0  PF=1
```

cf. 32 bit Flag Info
 → pushF (16-bit)
 → pushFd (32-bit) → prf



0 % ▾

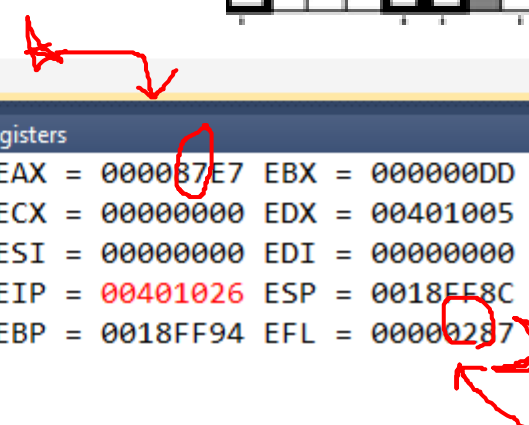
Memory 1

ddress: 0x00401000

```
<00401000 cc cc cc cc cc e9 06 00 00 00 cc cc cc cc cc c
<0040101D f6 02 d8 9f e8 1e 02 00 00 b8 00 00 00 00 bb 0
<0040103A c3 e8 04 02 00 00 b8 00 00 00 00 bb 00 00 00 0
<00401057 00 00 6a 00 e8 f4 0f 00 00 cc cc cc cc cc cc c
```

Registers

```
EAX = 000087E7 EBX = 000000DD
ECX = 00000000 EDX = 00401005
ESI = 00000000 EDI = 00000000
EIP = 00401026 ESP = 0018FF8C
EBP = 0018FF94 EFL = 00000287
```



Flags: OF – for signed number

- **OF (Overflow flag)**

- OF=1 when there is a CARRY IN but no CARRY OUT
- OF=1 when there is a CARRY OUT but no CARRY IN
- If OF=1, result is wrong when adding 2 **signed numbers**

- **Example**

483F + 645A → AC99

Carry In but no Carry Out

→ OF=1

No Carry Out → CF=0

1 Carry In

↖
0100 1000 0011 1111
0110 0100 0101 1010 +

1010 1100 1001 1001

- **Interpretation:**

- If the operation is for unsigned number addition → Correct
- If the operation is for signed numbers → Incorrect

Flags1.asm

IF: Interrupt Enable Flag
TF: Trap Flag

```
; (2)
mov EAX,0
mov EBX,0
mov AX, 483Fh
mov BX, 645Ah
add AX,BX ; AX = 483F + 645A
call DumpRegs

; (3)
mov EAX,0
mov EBX,0
mov EAX, 0FFFFFF97h
add EAX, 158 ; EAX = EAX + 158
call DumpRegs

exit
```

```
EAX=0000AC99 EBX=0000645A ECX=0013FFB0 EDX=7C90E514
ESI=0FA7F9D0 EDI=00000020 EBP=0013FFF0 ESP=0013FFC4
EIP=00401040 EFL=00000A96 CF=0 SF=1 ZF=0 OF=1 AF=1 PF=1
```

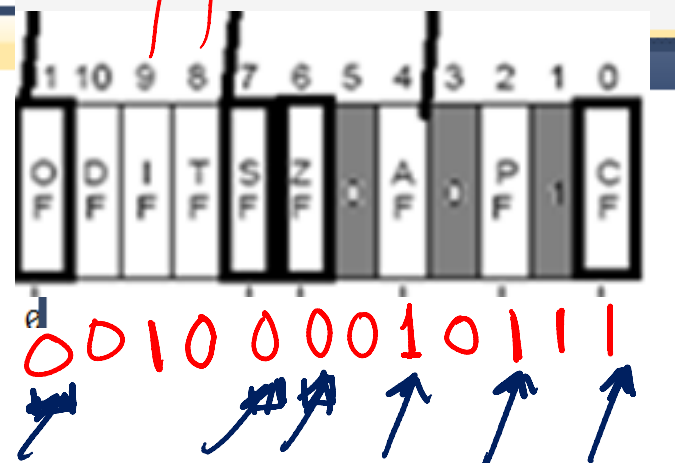
```
EAX=00000035 EBX=00000000 ECX=0013FFB0 EDX=7C90E514
ESI=0FA7F9D0 EDI=00000020 EBP=0013FFF0 ESP=0013FFC4
EIP=00401059 EFL=00000217 CF=1 SF=0 ZF=0 OF=0 AF=1 PF=1
```

main ENDP

Registers

```
EAX = 00000035 EBX = 00000000 ECX = 0013FFB0
EDX = 7C90E514 ESI = 0FA7F9D0 EDI = 00000020
EIP = 00401059 ESP = 0013FFC4 EBP = 0013FFF0
EFL = 00000217
```

00110101
Even # of 1's



Details of the Second Case

$$\begin{array}{r} \text{FFFFF} \boxed{\text{FF97}} \\ \text{9E} \end{array}$$

$$1,00000035 \quad 21 = \underline{16} + \underline{5}$$

$$\rightarrow 15$$

Carry out \rightarrow \leftarrow Carry in

$$\begin{array}{r} \text{①} \text{①} // // // // // \\ // // // // 1001 0111 \\ 1001 1110 \end{array}$$

$$9 + 9 + 1 = 19 \Rightarrow 16 + 3$$

$$\rightarrow \underline{1} \underline{3}$$

$$\begin{array}{r} // // // // 1011 0101 \end{array}$$

Carry from 3rd to 4th bit pos.

AF

SUB (Subtract Integers)

- **SUB:**

- Operation: $(DST - SRC) \rightarrow DST$
- subtracts the source operand from the destination operand and replaces the destination operand with the result. **If a borrow is required, the CF is set.** The operands may be signed or unsigned bytes, words, or doublewords.

- **label mnemonic dst, src**

EAX: 00 00 00 75 sub ecx, eax
ECX: 00 00 01 A2

[ECX] - [EAX] ----> [ECX]

EAX	00	00	00	75
ECX	00	00	01	2D

SF 0 ZF 0 CF 0 OF 0

SUB (Subtract Integers) – Manual Check

ECX - EAX

ECX 00 00 01 A2
-) EAX 00 00 00 75

	ECX	00	00	01	A2
+) 16's Complement of EAX		FF	FF	FF	8B
		<hr/>			
		00	00	01	2D

For Flag Check --> Binary

1 ←	0000	0000	0000	0000	0000	0001	1010	0010
	1111	1111	1111	1111	1111	1111	1000	1011
←							01	01010101

Carry In & Carry Out --> OF=0

CF=0 because Carry Out (*Note - subtraction)

ADD & SUB Examples – Class Activity

SF: Sign Flag
ZF: Zero Flag
CF: Carry Flag
OF: Overflow Flag

Fill the blanks for the registers and the Flags

GroupNO

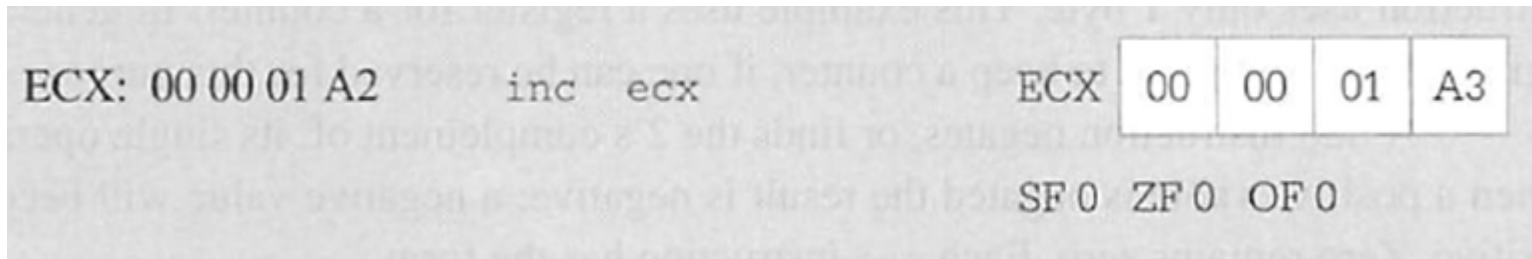
Names:

Instruction	EAX	EBX	ECX	CF	SF	ZF	OF
mov EAX, 75h							
mov ECX, 1A2h							
sub EAX, ECX							
mov AX, 77ACh							
mov CX, 4B35h							
add AX, CX							
mov EAX, 75h							
mov ECX, 1A2h							
sub ECX, EAX							
mov EBX, 0							
mov BL, 4Bh							
add BL, 4							
mov BX, 0FF20h							
sub BX, 0FF20h							
mov EAX, 9							
add EAX, 1							
mov EBX, 100h							
sub EBX, 1							

INC & DEC

- INC (Increment)

- $DST + 1 \rightarrow DST$
- adds one to the destination operand. **INC does not affect CF.** Use ADD with an immediate value of 1 if an increment that updates carry (CF) is needed.

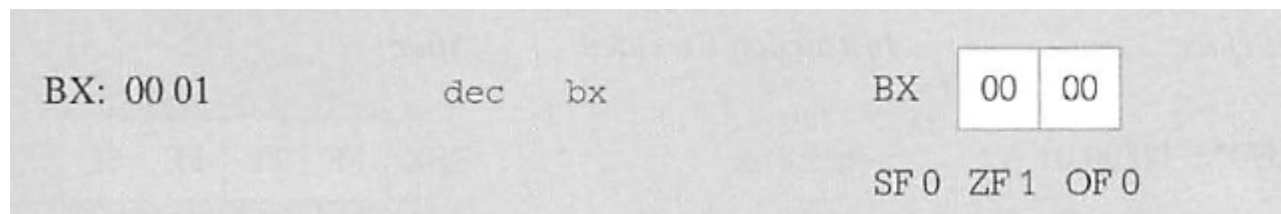


ECX: 00 00 01 A2 inc ecx ECX 00 00 01 A3

SF 0 ZF 0 OF 0

- DEC (Decrement)

- $DST - 1 \rightarrow DST$
- subtracts 1 from the destination operand. **DEC does not update CF.** Use SUB with an immediate value of 1 to perform a decrement that affects carry.



BX: 00 01 dec bx BX 00 00

SF 0 ZF 1 OF 0

CMP and NEG

- CMP (Compare)
 - DST – SRC
 - **subtracts** the source operand from the destination operand. **It updates OF, SF, ZF, AF, PF, and CF** but **does not alter the source and destination operands.**

```
cmp    eax, 356
cmp    wordOp, 0d3a6h
cmp    bh, '$'
```

	0	1	2	3	4	5	6	7
0	NUL	DLE	space	0	@	P	`	p
1	SOH	DC1 XON		1	A	Q	a	q
2	STX	DC2	"	2	B	R	b	r
3	ETX	DC3 XOFF	#	3	C	S	c	s
4	EOT	DC4	\$	4	D	T	d	t
5	ENQ	NAK	%	5	E	U	e	u
6	ACK	SYN	&	6	F	V	f	v
7	BEL	ETB	'	7	G	W	g	w

- NEG (Negate)
 - 0 – DST → DST
 - **subtracts a signed integer operand from zero.** The effect of NEG is to **reverse the sign** of the operand from positive to negative or from negative to positive (i.e., **16's complement**)
 - **SF and ZF are affected**

```
EBX: 00 00 01 A2    neg    ebx    EBX  FF  FF  FE  5E
                                     SF 1  ZF 0
```

Inc/Neg Practice

IncNeg.asm

```
mov EDX,OFFSET prompt
call WriteString

call ReadDec
mov z, EAX

mov EAX, x
add EAX, y
mov EBX, z
add EBX, EBX
call DumpRegs
sub EAX, EBX
call DumpRegs
inc EAX
call DumpRegs
neg EAX
call DumpRegs

exit
main ENDP
```

```
INCLUDE Irvine32.inc

.data
prompt    BYTE "Enter your number: ",0
x         DWORD ?
y         DWORD ?
z         DWORD ?

.code
main PROC
    mov EAX,0
    mov EBX,0

    mov EDX,OFFSET prompt
    call WriteString

    call ReadDec
;Others -- ReadHex (Hex number), ReadInt (signed number)
    mov x,EAX
```

100 %

Memory1

Address: 0x00405000

0x00405000	45 6e 74 65 72 20 79 6f	Enter yo
0x00405008	75 72 20 6e 75 6d 62 65	ur numbe
0x00405010	72 3a 20 20 00 23 00 00	r: .#..
0x00405018	00 2f 00 00 00 1a 00 00	./.....
0x00405020	00 00 00 00 00 00 00 00

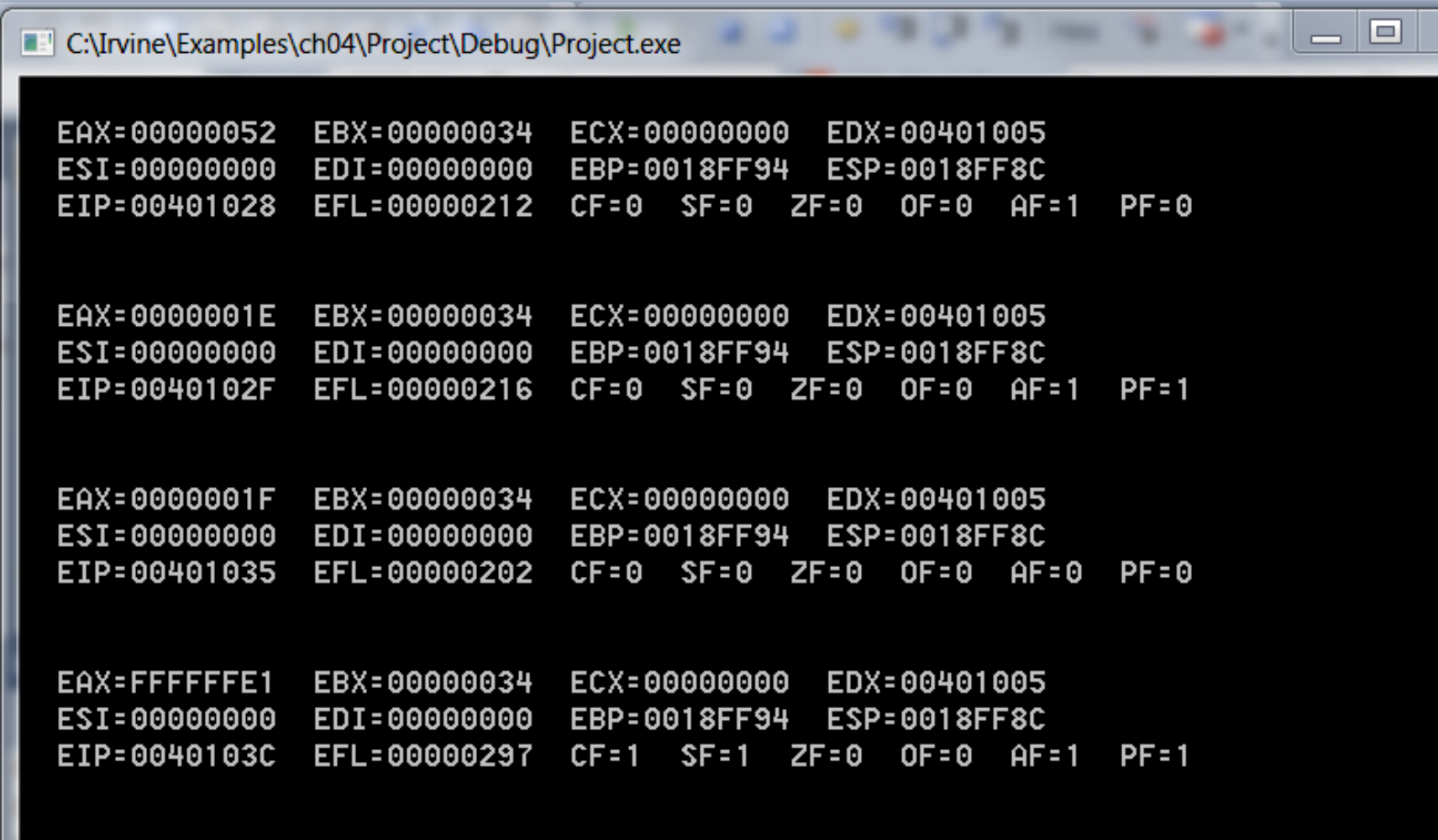
Registers

EAX = 00000052	EBX = 00000034
ECX = 00000000	EDX = 00405000
ESI = 00000000	EDI = 00000000
EIP = 0040106E	ESP = 0018FF8C
EBP = 0018FF94	EFL = 00000212

Link Library Procedures – Just a few

- DumpRegs
 - Displays EAX, EBX, etc
- ReadDec
 - Reads a 32-bit unsigned decimal integer from keyboard and returns the value in EAX
- ReadHex
 - Reads a 32-bit unsigned hex integer from the keyboard and returns the value in EAX
- ReadInt
 - Reads a 32-bit signed decimal integer from the keyboard and returns the value in EAX
- WriteString
 - Write a null-terminated string to the console window (*Pass the string's offset in EDX)

DumpRegs



```
C:\Irvine\Examples\ch04\Project\Debug\Project.exe

EAX=00000052  EBX=00000034  ECX=00000000  EDX=00401005
ESI=00000000  EDI=00000000  EBP=0018FF94  ESP=0018FF8C
EIP=00401028  EFL=00000212  CF=0  SF=0  ZF=0  OF=0  AF=1  PF=0

EAX=0000001E  EBX=00000034  ECX=00000000  EDX=00401005
ESI=00000000  EDI=00000000  EBP=0018FF94  ESP=0018FF8C
EIP=0040102F  EFL=00000216  CF=0  SF=0  ZF=0  OF=0  AF=1  PF=1

EAX=0000001F  EBX=00000034  ECX=00000000  EDX=00401005
ESI=00000000  EDI=00000000  EBP=0018FF94  ESP=0018FF8C
EIP=00401035  EFL=00000202  CF=0  SF=0  ZF=0  OF=0  AF=0  PF=0

EAX=FFFFFFE1  EBX=00000034  ECX=00000000  EDX=00401005
ESI=00000000  EDI=00000000  EBP=0018FF94  ESP=0018FF8C
EIP=0040103C  EFL=00000297  CF=1  SF=1  ZF=0  OF=0  AF=1  PF=1
```

Inc Dec Cmp Neg Examples – Class Activity

INC DEC NEG Class Activity

Fill the blanks for the registers and the Flags

GroupNO

Names:

Instruction	EAX	EBX	ECX	CF	SF	ZF	OF
mov ECX, 1A2h							
inc ECX							
mov EAX, 0F5h							
dec AL							
mov EBX, 1							
dec BX							
mov ECX, 7FFFFFFFh							
inc ECX							
mov EBX, 1A2h							
neg BX							
mov EAX, 0F500h							
neg AH							
mov EAX, 0							
neg EAX							

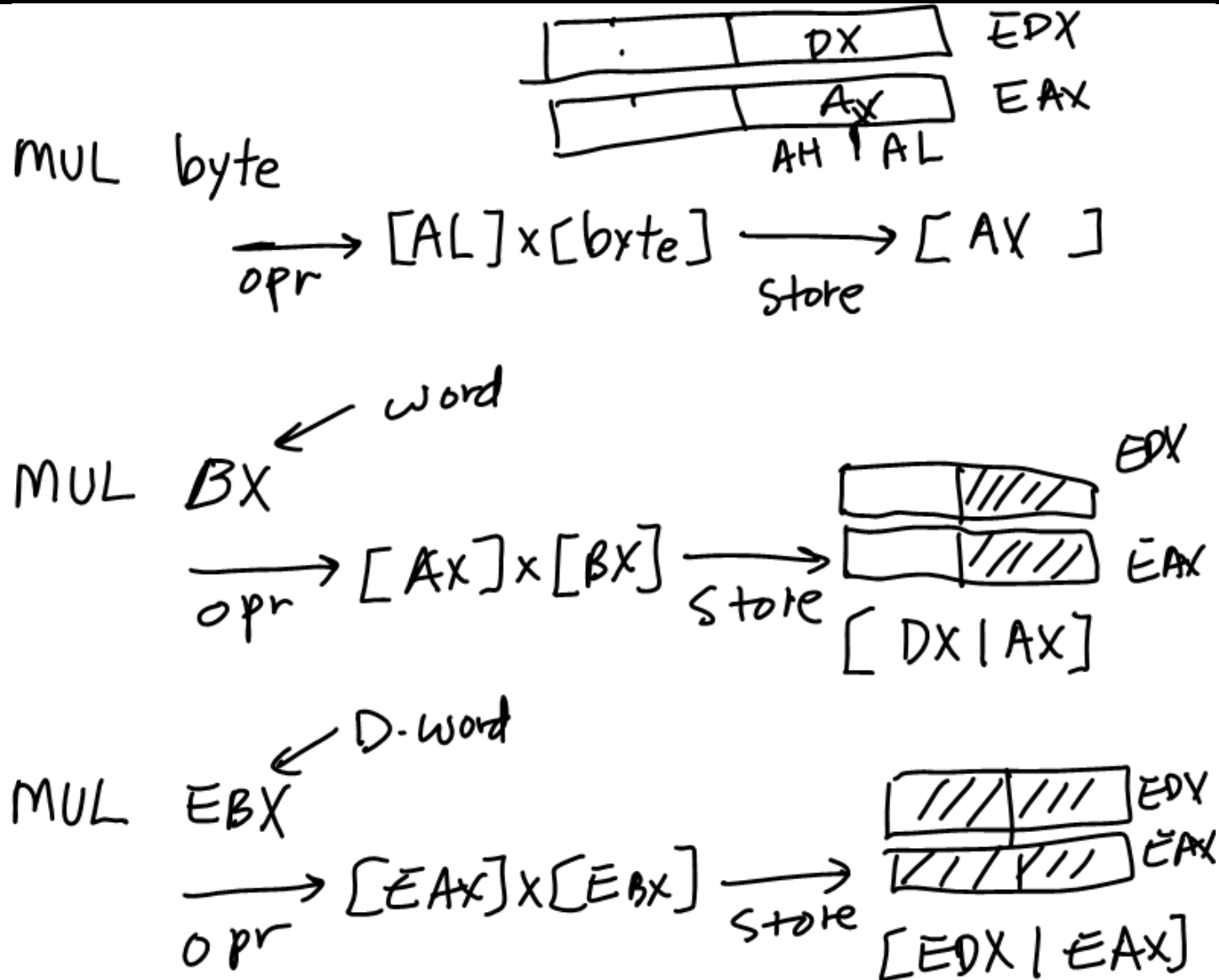
Multiplication Instruction - MUL

Syntax:

MUL *Source* → E (A?) ^L x ^X *Source*

- MUL (Unsigned Integer Multiply)
 - performs an unsigned multiplication of the source operand and the **accumulator [(E)AX]**.
 - **If the source is a byte**, the processor multiplies it by the contents of **AL** and returns the **double-length** result to **AH and AL (Concatenated)** i.e, **AX**.
 - **If the source operand is a word**, the processor multiplies it by the contents of **AX** and returns the **double-length** result to **DX and AX**.
 - **If the source operand is a double-word**, the processor multiplies it by the contents of **EAX** and returns the **64-bit result** in **EDX and EAX (Concatenated)**. MUL sets CF and OF when the upper half of the result is nonzero; otherwise, they are cleared.
 - Operand **cannot** be immediate

MUL Opr/Store Summary



MUL - Exercise

Note Title
9/27/2011

double word

↓

EAX 00 00 00 05

EBX 00 00 00 02

mul ebx →

00 00 00 00 EDX

00 00 00 0A EAX

EAX xx xx 00 05

EBX xx xx 00 02

EDX xx xx xx xx

mul bx →

xx xx 00 00 EDX

xx xx 00 0A EAX

EAX 00 00 00 0A

EDX xx xx xx xx

mul eax →

00 00 00 00 EDX

00 00 00 64 EAX

EAX xx xx xx 05

factor ← byte (FF)
(mem loc.)

mul factor →

~~xx~~ xx 04FB EAX

MUL - Exercise

EAX XX XX XX 05 $\xrightarrow{\text{mul factor}}$ ~~XX~~ XX 04FB EAX
 factor ← byte (FF)
 (mem loc.)

FF
x 5

1
16
32
48
64
80
96

carry Carry

5x15 = 75 --> 64 + 11 --> 4Bh

5x15+4 = 79 --> 64 + 15 --> 4Fh

FF
x 5
 4FB

IMUL (Signed Integer Multiply)

- performs a signed multiplication operation. IMUL has three variations:
 - 1. An **one-operand form**. The operand may be a byte, word, or doubleword located in memory or in a general register. This instruction uses EAX and EDX as implicit operands in the same way as the MUL instruction.

`imul source`

- 2. A two-operand form. One of the source operands may be in any general register while the other may be either in memory or in a general register. The product replaces the general-register operand.

`imul destination register, source`

- The immediate operand is treated as **signed**. If the immediate operand is a byte, the processor **automatically sign-extends to the size of storage (destination)** before performing the multiplication.

IMUL

EAX 00 00 00 05
EBX 00 00 00 02

imul ebx →

EDX 00 00 00 00
EAX 00 00 00 0A

EAX XXXX 00 05
EBX XX XX 00 02

imul bx →

EDX XX XX 00 00
EAX XX XX 00 0A

EAX XXXX XX 05
factor (FF)

imul factor →

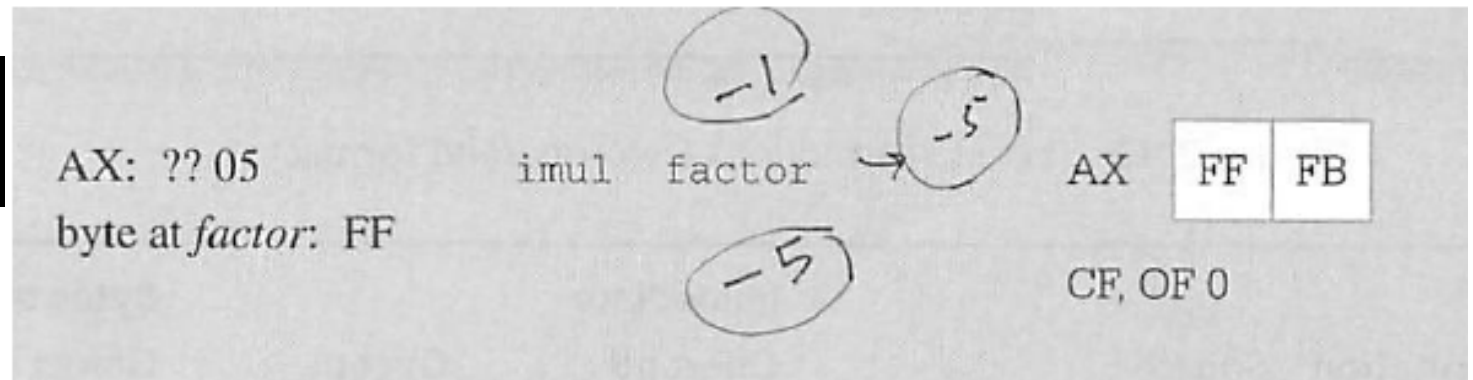
EAX XX XX FF FB

EAX 00 00 00 0A

imul ebx, 10 →

EBX 00 00 00 64

IMUL



Sign-ext
to the size
of DST (i.e. AX)

FFFF
x 0005

1
16
32
48
64
80
96

carry Carry

5x15 = 75 --> 64 + 11 --> 4Bh

5x15+4 = 79 --> 64 + 15 --> 4Fh

5x15+4 = 79 --> 64 + 15 --> 4Fh

5x15+4 = 79 --> 64 + 15 --> 4Fh

FFFF
x 0005
FFFB

Division Instruction

Syntax:

Q R

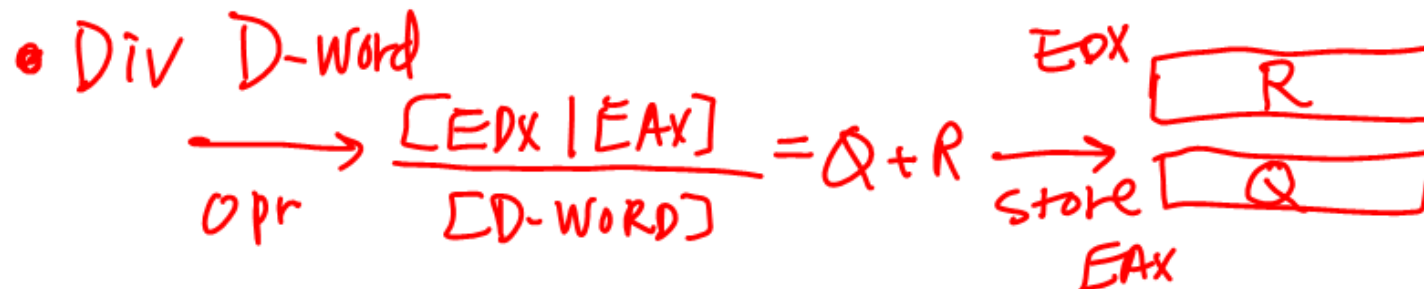
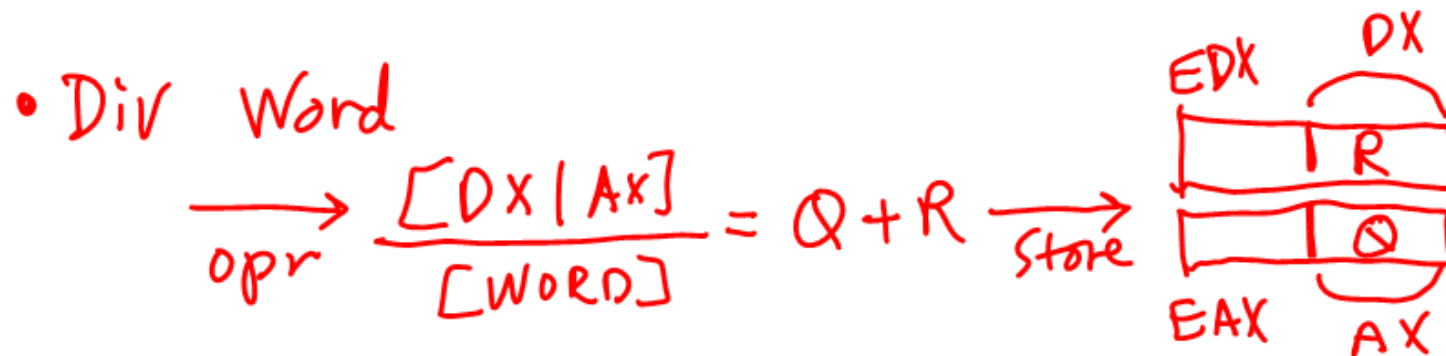
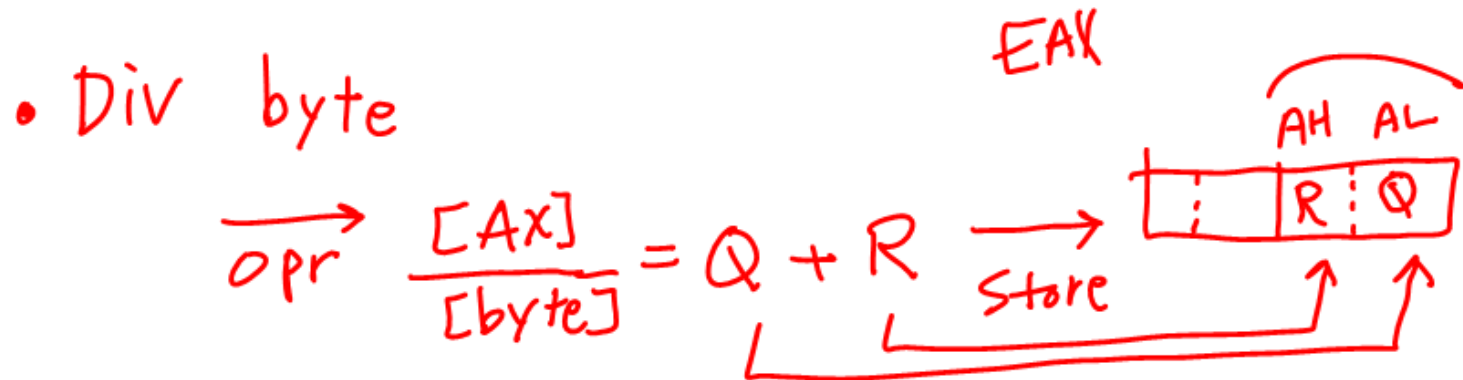
DIV Divider  E (A?)/Divider

- DIV (Unsigned Integer Divide)
 - performs an unsigned division of the accumulator by the source operand.
 - The dividend (the accumulator) is twice the size of the divisor (the source operand)

–	Size of Source Operand (divisor)	Dividend	Quotient	Remainder
	Byte	AX	AL	AH
	Word	DX:AX	AX	DX
	Doubleword	EDX:EAX	EAX	EDX

- IDIV (Signed Integer Divide)
 - performs a signed division of the accumulator by the source operand.
 - uses the same registers as the DIV instruction

DIV opr/store summary



DIV & IDIV

$100/13$
 EDX 00 00 00 00
 EAX 00 00 00 64
 EBX 00 00 00 0D
 $\xrightarrow{\text{div ebx}}$

EDX 0000 00 09 R
 EAX 0000 00 07 Q

EAX 00 00 00 64
 divisor (byte) 0D
 $\xrightarrow{\text{div divisor}}$

EAX 00 00 09 07
 R Q

$100/-13$
 EDX 00 00 00 00
 EAX 00 00 00 64
 ECX FF FF FF F3
 $\xrightarrow{\text{idiv ecx}}$

EDX 00 00 00 09 R
 EAX FF FF FF F9 Q

$-100/13$
 EDX FFFFFFFF
 EAX FFFFFFFC
 ECX 00 00 00 0D
 $\xrightarrow{\text{idiv ecx}}$

EDX FFFFFFFF R
 EAX FFFFFFF9 Q

IDIV cases

dividing Same sign

$100 / -13$	$100 = (-13) \cdot Q + R$	-7	9
$-100 / 13$	$-100 = (13) \cdot Q + R$	-7	-9
$-100 / -13$	$-100 = (-13) \cdot Q + R$	7	-9
$100 / 13$	$100 = (13) \cdot Q + R$	7	9

DIV vs IDIV

EAX 00 00 FE 01
 EBX 00 00 00 E0

65025
 -511
 224
 -32

$290 = 256 + 32 + 2$

div bL

$65025/224 = 290 + 65$
 Q R

idiv bL

EAX 00 00 E1 0F
 R Q
 -31 15

$-511/32$
 $= 15 - 31$
 Q R

$-31 \rightarrow -1F \rightarrow E1$
 d H 2's

$$\begin{aligned}
 -511 &= (-32) * Q + R \\
 &= (-32) * 15 + (-31)
 \end{aligned}$$

$$-32 * 15 = -480$$

$$-32 * 16 = -512$$

```

>>> 65025/224
290
>>> 65025-224*290
65
>>>
>>> hex(290)
'0x122'
>>> hex(65)
'0x41'
  
```

Mul Div --- Class Activity

```

.code
main PROC
; (A) MUL & IMUL
; (1)
        mov     EAX, 0E4h
        mov     EBX, 2
        mul     EBX
; (2)
        mov     EAX, 0FFh
        mov     EDX, 0FFFF0002h
        mul     AX
; (3)
        mov     EAX, 0FF0Fh
        mov     EBX, 0FFFF4C02h
        mov     EDX, 0FFFF0002h
        mul     BH
; (4)
        mov     EAX, 17h
        mov     ECX, 0B2h
        mov     EDX, 0FFFF0002h
        imul    ECX
; (5)
        mov     EAX, 0FF0Fh
        mov     EBX, 0FFFF4C02h
        mov     EDX, 0FFFF0002h

```

Names :

[illegible]

Mul Div --- Class Activity

```

        imul     BH

; (B) DIV & IDIV

; (1)
        mov     EAX, 9Ah
        mov     EBX, 0Fh
        mov     EDX, 0
        idiv    EBX

; (2)
        mov     EAX, 0FF75h
        idiv    COUNT

; (3)
        mov     EAX, 0FFFFFFF9Ah
        mov     ECX, 0FFFFFFFC7h
        mov     EDX, 0FFFFFFFFFh
        idiv    ECX

; (4)
        mov     EAX, 9Ah
        mov     ECX, 0FFC7h
        mov     EDX, 0
        idiv    CX

; (5)
        mov     EAX, 75h
        div     COUNT

        exit

main ENDP
END main

```

[illegible]

MulDiv.asm

MulDiv.asm X

TITLE MUL and DIV Examples (MulDiv.asm)

INCLUDE Irvine32.inc

.data

COUNT BYTE 0FCh

.code

main PROC

; (A) MUL & IMUL

; (1)

mov EAX, 0E4h

mov EBX, 2

mul EBX ; EAX? EDX?

call DumpRegs

; (2)

mov EAX, 0FFh

mov EDX, 0FFFF0002h

mul AX ; EAX? EDX?

call DumpRegs

; (3)

mov FAX, 0FF0Fh

100 %

Memory1

Address: 0x0018FF80

0x0018FF80	46	02	00	00	c8	01	00	00	21	10	40	00	8a	33	af	7
0x0018FF9D	e0	fd	7e	db	0a	12	77	00	00	00	00	00	00	00	00	0
0x0018FFBA	00	00	a0	ff	18	00	00	00	00	ff	ff	ff	ff	f5	7	
0x0018FFD7	00	45	9f	11	77	05	10	40	00	00	e0	fd	7e	00	00	0
0x0018FFF4	05	10	40	00	00	e0	fd	7e	00	00	00	00	41	63	74	7

Registers

EAX = 000001C8	EBX = 00000002
ECX = 00000000	EDX = 00000000
ESI = 00000000	EDI = 00000000
EIP = 00401021	ESP = 0018FF8C
EBP = 0018FF94	EFL = 00000246

MulDiv.asm

MulDiv.asm

```
; (4)
mov  EAX, 9Ah
mov  ECX, 0FFC7h
mov  EDX, 0
idiv CX          ;EAX?  EDX?

; (5)
; mov  EAX, 0FF75h
; div  COUNT     ;EAX?

exit
```

Memory 1

Address: 0x00405031

0x00405031	00 00 00 00 00 00 00 00 00 00
0x0040503B	00 00 00 00 00 00 00 00 00 00
0x00405045	00 00 00 00 00 00 00 00 00 00
0x0040504F	00 00 00 00 00 00 00 00 00 c8È

Registers

EAX = 0000FFFE	EBX = 0000000F	ECX = 0000FFC7
EDX = 00000028	ESI = 00000000	EDI = 00000000
EIP = 0040109B	ESP = 0018FF8C	EBP = 0018FF94
EFL = 00000A07		

DumpRegs

```
1  EAX=0000000A  EBX=0000000F  ECX=000000B2  EDX=00000004  
   ESI=00000000  EDI=00000000  EBP=0018FF94  ESP=0018FF8C  
   EIP=0040108B  EFL=00000A47  CF=1   SF=0   ZF=1   OF=1   AF=0   PF=1  
  
2  EAX=0000FD22  EBX=0000000F  ECX=000000B2  EDX=00000004  
   ESI=00000000  EDI=00000000  EBP=0018FF94  ESP=0018FF8C  
   EIP=0040109B  EFL=00000A47  CF=1   SF=0   ZF=1   OF=1   AF=0   PF=1  
  
3  EAX=00000001  EBX=0000000F  ECX=FFFFFFC7  EDX=FFFFFFD3  
   ESI=00000000  EDI=00000000  EBP=0018FF94  ESP=0018FF8C  
   EIP=004010B1  EFL=00000A47  CF=1   SF=0   ZF=1   OF=1   AF=0   PF=1  
  
4  EAX=0000FFFE  EBX=0000000F  ECX=0000FFC7  EDX=00000028  
   ESI=00000000  EDI=00000000  EBP=0018FF94  ESP=0018FF8C  
   EIP=004010C8  EFL=00000A47  CF=1   SF=0   ZF=1   OF=1   AF=0   PF=1
```

TcTf.asm

```

TcTf.asm - Notepad
File Edit Format View Help
TITLE Tc to Tf Conversion Example      (TcTf.asm)

INCLUDE Irvine32.inc
.data
prompt BYTE "Enter your temperature in Celcius (Tc): ",0
TC      DWORD ?
TF      DWORD ?

.code
main PROC
    mov     EAX,0
    mov     EBX,0

    mov     EDX,OFFSET prompt
    call    WriteString

    call    ReadDec
;Others -- ReadHex (Hex number), ReadInt (signed number)
    mov     TC,EAX

    mov     EAX, TC
    imul    EAX, 9
    add     EAX, 2
    mov     EBX, 5
    cdq
    idiv    EBX
    add     EAX, 32
    mov     TF, eax

    exit
main ENDP
END main

```

32

Integer

$$\frac{5}{2} \Rightarrow 2$$

Round off

$$\frac{5+1}{2} \Rightarrow 3$$

$\frac{1}{2}$ of divisor

```

>>> 5/2
2
>>> 1/2
0
>>> (5+1)/2
3
>>> 9/5
1
>>> 9/5.
1.8
>>> (9+5/2)/5
2
>>>

```

$\frac{Q}{EAX}$ $\frac{R}{EDX}$

TcTf.asm - Debugging

```
mov EDX,OFFSET prompt
call WriteString

call ReadDec
;Others -- ReadHex (Hex number), ReadInt (signed number)
mov TC,EAX

mov EAX, TC
imul EAX, 9
add EAX,2
mov EBX,5
cdq
idiv EBX
add EAX,32
mov TF,eax

exit
main ENDP
END main
```

$$T_{(^{\circ}\text{F})} = T_{(^{\circ}\text{C})} \times 9/5 + 32$$

100 %

Memory1

Address: 0x00405000

0x00405000	45 6e 74 65 72 20 79 6f	Enter yo
0x00405008	75 72 20 74 65 6d 70 65	ur tempe
0x00405010	72 61 74 75 72 65 20 69	rature i
0x00405018	6e 20 43 65 6c 63 69 75	n Celciu
0x00405020	73 20 28 54 63 29 3a 20	s (Tc):
0x00405028	20 00 20 00 00 00 5a 00Z.

Registers

EAX = 0000005A	EBX = 00000005
ECX = 00000000	EDX = 00000000
ESI = 00000000	EDI = 00000000
EIP = 00401049	ESP = 0018FF8C
EBP = 0018FF94	EFL = 00000206

ASM Coding Assignment

- **Individual Work**
- Write an ASM code which converts a temperature in Fahrenheit to a temperature in Celsius.
 - Should read a temperature from the keyboard
 - A report
 - Code
 - Debugging results
 - Screen captures
 - Softcopy (Docx, pdf) submission (via email): Due: 11:59pm, Sunday Nov 13, 2016.
 - File naming convention: **lastname_ASM_1.docx** or **lastname_ASM_1.pdf**