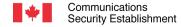
# **General Operations**

Because, in the end, you're still to write something useful.

© Government of Canada

This document is the property of the Government of Canada. It shall not be altered, distributed beyond its intended audience, produced, reproduced or published, in whole or in any substantial part thereof, without the express permission of CSE.





#### **ALU instructions**

Instruction	Example	Description
add	add eax, 0x04	Add 4 to the content of eax.
and	and eax, 0x01	Do a binary <b>and</b> between eax and 0x01.
or	or eax, 0xFF	Do a binary <b>or</b> between eax and 0xFF.
sub	sub eax, 0x01	Subtract 0x01 from eax.
test	test eax, eax	This is equivalent to cmp and does not change the register values.
xor	xor eax, eax	Do a binary <b>xor</b> between eax and eax.



#### **SHIFT** instructions

Instruction	Example	Description
sal	sal eax, 0x01	Proceed to a left shift while keeping the sign of the data in eax
sar	sar eax, 0x01	Same as sal but shift is done to the right.
shl	shl eax, 0x01	Proceed to a left shift. The sign bit is not kept.
shr	shr eax, 0x01	Same as shI but sift is done to the left.
rol	rol eax, 0x01	Rotate the bits to the left.
ror	ror eax, 0x01	Rotate the bits to the right.





#### Shift arithmetic VS. Shift VS. Rotation

**Shift arithmetic** (on a byte):

Let's pretend we have the following value

0b0000001

After applying a 1 bit left shift the value will be:

0b00000010

Had the value been **0b1000001** the new value would have been

0b10000010

Communications
Security Establishment

Centre de la sécurité des télécommunications

Shift (on a byte):

Let's pretend we have the following value

0b0000001

After applying a 1 bit left shift the value will be:

0b00000010

Had the value been 0b10000001 the new value would have been

0b00000010

Rotation (on a byte):

Let's pretend we have the following value

0b0000001

After applying a 1 bit left rotation the value will be:

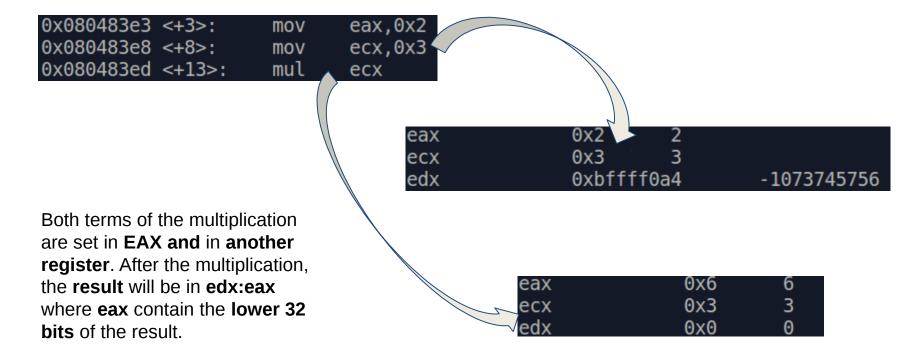
0b0000010

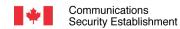
Had the value been 0b10000001 the new value would have been

0b00000011



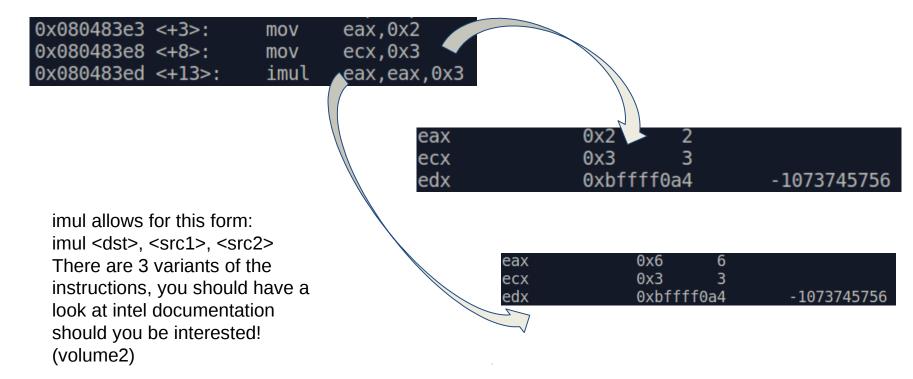
### Multiplication - unsigned (mul)







### **Multiplication - signed (imul)**







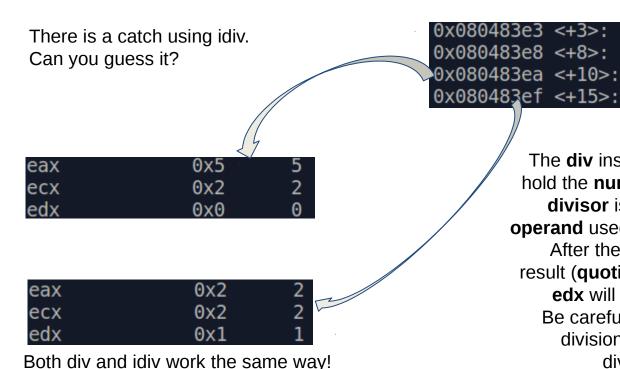
eax,0x5

edx,edx

ecx,0x2

ecx

### Division - unsigned (div) AND signed (idiv)



The **div** instruction uses **edx:eax** to hold the **number to be divided**. The divisor is then determined by the **operand** used with the div instruction. After the div, eax will contain the result (quotient) of the division while edx will contain the remainder. Be careful!, AH:AL will be used for division on bytes and DX:AX for division on words (16 bits).

mov

xor

mov

div



PAGE 7



## Lets write some code!



