IMUL—Signed Multiply

Opcode	Instruction	64-Bit Mode	Compat/ Leg Mode	Description
F6 /5	IMUL *r/m8**	Valid	Valid	AX← AL ∗ *r/m* byte.
F7 /5	IMUL *r/m16*	Valid	Valid	DX:AX ← AX ∗ *r/m* word.
F7 /5	IMUL *r/m32*	Valid	Valid	EDX:EAX ← EAX ∗ *r/m32*.
REX.W + F7 /5	IMUL *r/m64*	Valid	N.E.	RDX:RAX ← RAX ∗ *r/m64*.
0F AF /*r*	IMUL *r16, r/m16*	Valid	Valid	word register ← word register ∗ *r/m16*.
0F AF /*r*	IMUL *r32, r/m32*	Valid	Valid	doubleword register ← doubleword register ∗ *r/m32.*
REX.W + 0F AF /*r*	IMUL *r64, r/m64*	Valid	N.E.	Quadword register ← Quadword register ∗ *r/m64*.
6B /*r ib*	IMUL *r16, r/m16, imm8*	Valid	Valid	word register ← *r/m16* ∗ sign-extended immediate byte.
6B /*r ib*	IMUL *r32, r/m32, imm8*	Valid	Valid	doubleword register ← *r/m32* ∗ sign-extended immediate byte.
REX.W + 6B /*r ib*	IMUL *r64, r/m64, imm8*	Valid	N.E.	Quadword register ← *r/m64* ∗ sign-extended immediate byte.
6B /*r ib*	IMUL *r16, imm8*	Valid	Valid	word register ← word register ∗ sign-extended immediate byte.
6B /*r ib*	IMUL *r32, imm8*	Valid	Valid	doubleword register ← doubleword register ∗ sign-extended immediate byte.
REX.W + 6B /*r ib*	IMUL *r64, imm8*	Valid	N.E.	Quadword register ← Quadword register ∗ sign-extended immediate byte.
69 /*r iw*	IMUL *r16, r/m16, imm16*	Valid	Valid	word register ← *r/m16* ∗ immediate word.
69 /*r id*	IMUL *r32, r/m32, imm32*	Valid	Valid	doubleword register ← *r/m32* ∗ immediate doubleword.
REX.W + 69 /*r id*	IMUL *r64, r/m64, imm32*	Valid	N.E.	Quadword register ← *r/m64* ∗ immediate doubleword.
69 /*r iw*	IMUL *r16, imm16*	Valid	Valid	word register ← *r/m16* ∗ immediate word.

Opcode	Instruction	64-Bit Mode	Compat/ Leg Mode	Description
69 /r *id*	IMUL *r32, imm32*	Valid	Valid	doubleword register ← *r/m32* * immediate doubleword.
REX.W + 69 /r *id*	IMUL *r64, imm32*	Valid	N.E.	Quadword register ← *r/m64* * immediate doubleword.

NOTES:

* In 64-bit mode, r/m8 can not be encoded to access the following byte registers if a REX prefix is used: AH, BH, CH, DH.

Description

Performs a signed multiplication of two operands. This instruction has three forms, depending on the number of operands.

- **One-operand form** — This form is identical to that used by the MUL instruction. Here, the source operand (in a general-purpose register or memory location) is multiplied by the value in the AL, AX, EAX, or RAX register (depending on the operand size) and the product is stored in the AX, DX:AX, EDX:EAX, or RDX:RAX registers, respectively.
- **Two-operand form** — With this form the destination operand (the first operand) is multiplied by the source operand (second operand). The destination operand is a general-purpose register and the source operand is an immediate value, a general-purpose register, or a memory location. The product is then stored in the destination operand location.
- **Three-operand form** — This form requires a destination operand (the first operand) and two source operands (the second and the third operands). Here, the first source operand (which can be a general-purpose register or a memory location) is multiplied by the second source operand (an immediate value). The product is then stored in the destination operand (a general-purpose register).

When an immediate value is used as an operand, it is sign-extended to the length of the destination operand format.

The CF and OF flags are set when significant bit (including the sign bit) are carried into the upper half of the result. The CF and OF flags are cleared when the result (including the sign bit) fits exactly in the lower half of the result.

The three forms of the IMUL instruction are similar in that the length of the product is calculated to twice the length of the operands. With the one-operand form, the product is stored exactly in the destination. With the two- and three- operand forms, however, the result is truncated to the length of the destination before it is stored in the destination register. Because of this truncation, the CF or OF flag should be tested to ensure that no significant bits are lost.

The two- and three-operand forms may also be used with unsigned operands because the lower half of the product is the same regardless if the operands are

signed or unsigned. The CF and OF flags, however, cannot be used to determine if the upper half of the result is non-zero.

In 64-bit mode, the instruction's default operation size is 32 bits. Use of the REX.R prefix permits access to additional registers (R8-R15). Use of the REX.W prefix promotes operation to 64 bits. Use of REX.W modifies the three forms of the instruction as follows.

- One-operand form —The source operand (in a 64-bit general-purpose register or memory location) is multiplied by the value in the RAX register and the product is stored in the RDX:RAX registers.
- **Two-operand form** — The source operand is promoted to 64 bits if it is a register or a memory location. If the source operand is an immediate, it is sign extended to 64 bits. The destination operand is promoted to 64 bits.
- **Three-operand form** — The first source operand (either a register or a memory location) and destination operand are promoted to 64 bits.

Operation

```
IF (NumberOfOperands = 1)
    THEN IF (OperandSize = 8)
        THEN
            AX ← AL * SRC  (* Signed multiplication *)
            IF AL = AX
                THEN CF ← 0; OF ← 0;
                ELSE CF ← 1; OF ← 1; FI;
        ELSE IF OperandSize = 16
            THEN
                DX:AX ← AX * SRC  (* Signed multiplication *)
                IF sign_extend_to_32 (AX) = DX:AX
                    THEN CF ← 0; OF ← 0;
                    ELSE CF ← 1; OF ← 1; FI;
            ELSE IF OperandSize = 32
                THEN
                    EDX:EAX ← EAX * SRC  (* Signed multiplication *)
                    IF EAX = EDX:EAX
                        THEN CF ← 0; OF ← 0;
                        ELSE CF ← 1; OF ← 1; FI;
                ELSE (* OperandSize = 64 *)
                    RDX:RAX ← RAX * SRC  (* Signed multiplication *)
                    IF RAX = RDX:RAX
                        THEN CF ← 0; OF ← 0;
                        ELSE CF ← 1; OF ← 1; FI;
                FI;
        FI;
```