

**FMUL/FMULP/FIMUL—Multiply**

Opcode	Instruction	64-Bit Mode	Compat/Leg Mode	Description
D8 /1	FMUL <i>m32fp</i>	Valid	Valid	Multiply ST(0) by <i>m32fp</i> and store result in ST(0).
DC /1	FMUL <i>m64fp</i>	Valid	Valid	Multiply ST(0) by <i>m64fp</i> and store result in ST(0).
D8 C8+i	FMUL ST(0), ST(i)	Valid	Valid	Multiply ST(0) by ST(i) and store result in ST(0).
DC C8+i	FMUL ST(i), ST(0)	Valid	Valid	Multiply ST(i) by ST(0) and store result in ST(i).
DE C8+i	FMULP ST(i), ST(0)	Valid	Valid	Multiply ST(i) by ST(0), store result in ST(i), and pop the register stack.
DE C9	FMULP	Valid	Valid	Multiply ST(1) by ST(0), store result in ST(1), and pop the register stack.
DA /1	FIMUL <i>m32int</i>	Valid	Valid	Multiply ST(0) by <i>m32int</i> and store result in ST(0).
DE /1	FIMUL <i>m16int</i>	Valid	Valid	Multiply ST(0) by <i>m16int</i> and store result in ST(0).

**Description**

Multiplies the destination and source operands and stores the product in the destination location. The destination operand is always an FPU data register; the source operand can be an FPU data register or a memory location. Source operands in memory can be in single-precision or double-precision floating-point format or in word or doubleword integer format.

The no-operand version of the instruction multiplies the contents of the ST(1) register by the contents of the ST(0) register and stores the product in the ST(1) register. The one-operand version multiplies the contents of the ST(0) register by the contents of a memory location (either a floating point or an integer value) and stores the product in the ST(0) register. The two-operand version, multiplies the contents of the ST(0) register by the contents of the ST(i) register, or vice versa, with the result being stored in the register specified with the first operand (the destination operand).

The FMULP instructions perform the additional operation of popping the FPU register stack after storing the product. To pop the register stack, the processor marks the ST(0) register as empty and increments the stack pointer (TOP) by 1. The no-operand version of the floating-point multiply instructions always results in the register stack being popped. In some assemblers, the mnemonic for this instruction is FMUL rather than FMULP.

The FIMUL instructions convert an integer source operand to double extended-precision floating-point format before performing the multiplication.

The sign of the result is always the exclusive-OR of the source signs, even if one or more of the values being multiplied is 0 or  $\infty$ . When the source operand is an integer 0, it is treated as a +0.

The following table shows the results obtained when multiplying various classes of numbers, assuming that neither overflow nor underflow occurs.

Table 3-29. FMUL/FMULP/FIMUL Results

SRC	DEST						
		$-\infty$	$-F$	$-0$	$+0$	$+F$	$+\infty$
	$-\infty$	$+\infty$	$+\infty$	*	*	$-\infty$	$-\infty$
	$-F$	$+\infty$	$+F$	$+0$	$-0$	$-F$	$-\infty$
	$-I$	$+\infty$	$+F$	$+0$	$-0$	$-F$	$-\infty$
	$-0$	*	$+0$	$+0$	$-0$	$-0$	*
	$+0$	*	$-0$	$-0$	$+0$	$+0$	*
	$+I$	$-\infty$	$-F$	$-0$	$+0$	$+F$	$+\infty$
	$+F$	$-\infty$	$-F$	$-0$	$+0$	$+F$	$+\infty$
	$+\infty$	$-\infty$	$-\infty$	*	*	$+\infty$	$+\infty$
	NaN	NaN	NaN	NaN	NaN	NaN	NaN

**NOTES:**

F Means finite floating-point value.

I Means Integer.

\* Indicates invalid-arithmetic-operand (#IA) exception.

This instruction's operation is the same in non-64-bit modes and 64-bit mode.

**Operation**

IF Instruction = FIMUL

THEN

DEST  $\leftarrow$  DEST \* ConvertToDoubleExtendedPrecisionFP(SRC);

ELSE (\* Source operand is floating-point value \*)

DEST  $\leftarrow$  DEST \* SRC;

FI;

IF Instruction = FMULP

THEN

PopRegisterStack;

FI;

**FPU Flags Affected**

- C1 Set to 0 if stack underflow occurred.  
Set if result was rounded up; cleared otherwise.
- C0, C2, C3 Undefined.

**Floating-Point Exceptions**

- #IS Stack underflow occurred.
- #IA Operand is an SNaN value or unsupported format.  
One operand is  $\pm 0$  and the other is  $\pm \infty$ .
- #D Source operand is a denormal value.
- #U Result is too small for destination format.
- #O Result is too large for destination format.
- #P Value cannot be represented exactly in destination format.

**Protected Mode Exceptions**

#GP(0)	If a memory operand effective address is outside the CS, DS, ES, FS, or GS segment limit. If the DS, ES, FS, or GS register is used to access memory and it contains a NULL segment selector.
#SS(0)	If a memory operand effective address is outside the SS segment limit.
#NM	CR0.EM[bit 2] or CR0.TS[bit 3] = 1.
#PF(fault-code)	If a page fault occurs.
#AC(0)	If alignment checking is enabled and an unaligned memory reference is made while the current privilege level is 3.
#UD	If the LOCK prefix is used.

**Real-Address Mode Exceptions**

#GP	If a memory operand effective address is outside the CS, DS, ES, FS, or GS segment limit.
#SS	If a memory operand effective address is outside the SS segment limit.
#NM	CR0.EM[bit 2] or CR0.TS[bit 3] = 1.
#UD	If the LOCK prefix is used.

**Virtual-8086 Mode Exceptions**

#GP(0)	If a memory operand effective address is outside the CS, DS, ES, FS, or GS segment limit.
#SS(0)	If a memory operand effective address is outside the SS segment limit.
#NM	CR0.EM[bit 2] or CR0.TS[bit 3] = 1.
#PF(fault-code)	If a page fault occurs.
#AC(0)	If alignment checking is enabled and an unaligned memory reference is made.
#UD	If the LOCK prefix is used.

**Compatibility Mode Exceptions**

Same exceptions as in protected mode.

**64-Bit Mode Exceptions**

#SS(0)	If a memory address referencing the SS segment is in a non-canonical form.
#GP(0)	If the memory address is in a non-canonical form.
#NM	CR0.EM[bit 2] or CR0.TS[bit 3] = 1.
#MF	If there is a pending x87 FPU exception.
#PF(fault-code)	If a page fault occurs.
#AC(0)	If alignment checking is enabled and an unaligned memory reference is made while the current privilege level is 3.
#UD	If the LOCK prefix is used.