

**FADD/FADDP/FIADD—Add**

Opcode	Instruction	64-Bit Mode	Compat/Leg Mode	Description
D8 /0	FADD <i>m32fp</i>	Valid	Valid	Add <i>m32fp</i> to ST(0) and store result in ST(0).
DC /0	FADD <i>m64fp</i>	Valid	Valid	Add <i>m64fp</i> to ST(0) and store result in ST(0).
D8 C0+i	FADD ST(0), ST(i)	Valid	Valid	Add ST(0) to ST(i) and store result in ST(0).
DC C0+i	FADD ST(i), ST(0)	Valid	Valid	Add ST(i) to ST(0) and store result in ST(i).
DE C0+i	FADDP ST(i), ST(0)	Valid	Valid	Add ST(0) to ST(i), store result in ST(i), and pop the register stack.
DE C1	FADDP	Valid	Valid	Add ST(0) to ST(1), store result in ST(1), and pop the register stack.
DA /0	FIADD <i>m32int</i>	Valid	Valid	Add <i>m32int</i> to ST(0) and store result in ST(0).
DE /0	FIADD <i>m16int</i>	Valid	Valid	Add <i>m16int</i> to ST(0) and store result in ST(0).

**Description**

Adds the destination and source operands and stores the sum in the destination location. The destination operand is always an FPU register; the source operand can be a 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 adds the contents of the ST(0) register to the ST(1) register. The one-operand version adds the contents of a memory location (either a floating-point or an integer value) to the contents of the ST(0) register. The two-operand version, adds the contents of the ST(0) register to the ST(i) register or vice versa. The value in ST(0) can be doubled by coding:

```
FADD ST(0), ST(0);
```

The FADDP instructions perform the additional operation of popping the FPU register stack after storing the result. 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 add instructions always results in the register stack being popped. In some assemblers, the mnemonic for this instruction is FADD rather than FADDP.)

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

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

When the sum of two operands with opposite signs is 0, the result is +0, except for the round toward  $-\infty$  mode, in which case the result is  $-0$ . When the source operand is an integer 0, it is treated as a +0.

When both operand are infinities of the same sign, the result is  $\infty$  of the expected sign. If both operands are infinities of opposite signs, an invalid-operation exception is generated. See Table 3-18.

Table 3-18. FADD/FADDP/FIADD Results

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

**NOTES:**

F Means finite floating-point value.

I Means integer.

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

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

**Operation**

IF Instruction = FIADD

THEN

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

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

DEST  $\leftarrow$  DEST + SRC;

FI;

IF Instruction = FADDP

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.  
Operands are infinities of unlike sign.
- #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 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.