

**AAA—ASCII Adjust After Addition**

Opcode	Instruction	Op/En	64-bit Mode	Compat/Leg Mode	Description
37	AAA	NP	Invalid	Valid	ASCII adjust AL after addition.

**Instruction Operand Encoding**

Op/En	Operand 1	Operand 2	Operand 3	Operand 4
NP	NA	NA	NA	NA

**Description**

Adjusts the sum of two unpacked BCD values to create an unpacked BCD result. The AL register is the implied source and destination operand for this instruction. The AAA instruction is only useful when it follows an ADD instruction that adds (binary addition) two unpacked BCD values and stores a byte result in the AL register. The AAA instruction then adjusts the contents of the AL register to contain the correct 1-digit unpacked BCD result.

If the addition produces a decimal carry, the AH register increments by 1, and the CF and AF flags are set. If there was no decimal carry, the CF and AF flags are cleared and the AH register is unchanged. In either case, bits 4 through 7 of the AL register are set to 0.

This instruction executes as described in compatibility mode and legacy mode. It is not valid in 64-bit mode.

**Operation**

```

IF 64-Bit Mode
  THEN
    #UD;
  ELSE
    IF ((AL AND 0FH) > 9) or (AF = 1)
      THEN
        AX ← AX + 106H;
        AF ← 1;
        CF ← 1;
      ELSE
        AF ← 0;
        CF ← 0;
    FI;
    AL ← AL AND 0FH;
  FI;

```

**Flags Affected**

The AF and CF flags are set to 1 if the adjustment results in a decimal carry; otherwise they are set to 0. The OF, SF, ZF, and PF flags are undefined.

**Protected Mode Exceptions**

#UD If the LOCK prefix is used.

**Real-Address Mode Exceptions**

Same exceptions as protected mode.

**Virtual-8086 Mode Exceptions**

Same exceptions as protected mode.

**Compatibility Mode Exceptions**

Same exceptions as protected mode.

**64-Bit Mode Exceptions**

#UD                      If in 64-bit mode.

**AAD—ASCII Adjust AX Before Division**

Opcode	Instruction	Op/En	64-bit Mode	Compat/Leg Mode	Description
D5 0A	AAD	NP	Invalid	Valid	ASCII adjust AX before division.
D5 <i>ib</i>	AAD <i>imm8</i>	NP	Invalid	Valid	Adjust AX before division to number base <i>imm8</i> .

**Instruction Operand Encoding**

Op/En	Operand 1	Operand 2	Operand 3	Operand 4
NP	NA	NA	NA	NA

**Description**

Adjusts two unpacked BCD digits (the least-significant digit in the AL register and the most-significant digit in the AH register) so that a division operation performed on the result will yield a correct unpacked BCD value. The AAD instruction is only useful when it precedes a DIV instruction that divides (binary division) the adjusted value in the AX register by an unpacked BCD value.

The AAD instruction sets the value in the AL register to  $(AL + (10 * AH))$ , and then clears the AH register to 00H. The value in the AX register is then equal to the binary equivalent of the original unpacked two-digit (base 10) number in registers AH and AL.

The generalized version of this instruction allows adjustment of two unpacked digits of any number base (see the "Operation" section below), by setting the *imm8* byte to the selected number base (for example, 08H for octal, 0AH for decimal, or 0CH for base 12 numbers). The AAD mnemonic is interpreted by all assemblers to mean adjust ASCII (base 10) values. To adjust values in another number base, the instruction must be hand coded in machine code (D5 *imm8*).

This instruction executes as described in compatibility mode and legacy mode. It is not valid in 64-bit mode.

**Operation**

```

IF 64-Bit Mode
  THEN
    #UD;
  ELSE
    tempAL ← AL;
    tempAH ← AH;
    AL ← (tempAL + (tempAH * imm8)) AND FFH;
    (* imm8 is set to 0AH for the AAD mnemonic.*)
    AH ← 0;
FI;

```

The immediate value (*imm8*) is taken from the second byte of the instruction.

**Flags Affected**

The SF, ZF, and PF flags are set according to the resulting binary value in the AL register; the OF, AF, and CF flags are undefined.

**Protected Mode Exceptions**

#UD If the LOCK prefix is used.

**Real-Address Mode Exceptions**

Same exceptions as protected mode.

**Virtual-8086 Mode Exceptions**

Same exceptions as protected mode.

**Compatibility Mode Exceptions**

Same exceptions as protected mode.

**64-Bit Mode Exceptions**

#UD                      If in 64-bit mode.

**AAM—ASCII Adjust AX After Multiply**

Opcode	Instruction	Op/En	64-bit Mode	Compat/Leg Mode	Description
D4 0A	AAM	NP	Invalid	Valid	ASCII adjust AX after multiply.
D4 <i>ib</i>	AAM <i>imm8</i>	NP	Invalid	Valid	Adjust AX after multiply to number base <i>imm8</i> .

**Instruction Operand Encoding**

Op/En	Operand 1	Operand 2	Operand 3	Operand 4
NP	NA	NA	NA	NA

**Description**

Adjusts the result of the multiplication of two unpacked BCD values to create a pair of unpacked (base 10) BCD values. The AX register is the implied source and destination operand for this instruction. The AAM instruction is only useful when it follows an MUL instruction that multiplies (binary multiplication) two unpacked BCD values and stores a word result in the AX register. The AAM instruction then adjusts the contents of the AX register to contain the correct 2-digit unpacked (base 10) BCD result.

The generalized version of this instruction allows adjustment of the contents of the AX to create two unpacked digits of any number base (see the “Operation” section below). Here, the *imm8* byte is set to the selected number base (for example, 08H for octal, 0AH for decimal, or 0CH for base 12 numbers). The AAM mnemonic is interpreted by all assemblers to mean adjust to ASCII (base 10) values. To adjust to values in another number base, the instruction must be hand coded in machine code (D4 *imm8*).

This instruction executes as described in compatibility mode and legacy mode. It is not valid in 64-bit mode.

**Operation**

```

IF 64-Bit Mode
    THEN
        #UD;
    ELSE
        tempAL ← AL;
        AH ← tempAL / imm8; (* imm8 is set to 0AH for the AAM mnemonic *)
        AL ← tempAL MOD imm8;
FI;

```

The immediate value (*imm8*) is taken from the second byte of the instruction.

**Flags Affected**

The SF, ZF, and PF flags are set according to the resulting binary value in the AL register. The OF, AF, and CF flags are undefined.

**Protected Mode Exceptions**

#DE If an immediate value of 0 is used.  
 #UD If the LOCK prefix is used.

**Real-Address Mode Exceptions**

Same exceptions as protected mode.

**Virtual-8086 Mode Exceptions**

Same exceptions as protected mode.

**Compatibility Mode Exceptions**

Same exceptions as protected mode.

**64-Bit Mode Exceptions**

#UD                      If in 64-bit mode.

**AAS—ASCII Adjust AL After Subtraction**

Opcode	Instruction	Op/En	64-bit Mode	Compat/Leg Mode	Description
3F	AAS	NP	Invalid	Valid	ASCII adjust AL after subtraction.

**Instruction Operand Encoding**

Op/En	Operand 1	Operand 2	Operand 3	Operand 4
NP	NA	NA	NA	NA

**Description**

Adjusts the result of the subtraction of two unpacked BCD values to create a unpacked BCD result. The AL register is the implied source and destination operand for this instruction. The AAS instruction is only useful when it follows a SUB instruction that subtracts (binary subtraction) one unpacked BCD value from another and stores a byte result in the AL register. The AAA instruction then adjusts the contents of the AL register to contain the correct 1-digit unpacked BCD result.

If the subtraction produced a decimal carry, the AH register decrements by 1, and the CF and AF flags are set. If no decimal carry occurred, the CF and AF flags are cleared, and the AH register is unchanged. In either case, the AL register is left with its top four bits set to 0.

This instruction executes as described in compatibility mode and legacy mode. It is not valid in 64-bit mode.

**Operation**

IF 64-bit mode

THEN

#UD;

ELSE

IF ((AL AND 0FH) > 9) or (AF = 1)

THEN

AX ← AX - 6;

AH ← AH - 1;

AF ← 1;

CF ← 1;

AL ← AL AND 0FH;

ELSE

CF ← 0;

AF ← 0;

AL ← AL AND 0FH;

FI;

FI;

**Flags Affected**

The AF and CF flags are set to 1 if there is a decimal borrow; otherwise, they are cleared to 0. The OF, SF, ZF, and PF flags are undefined.

**Protected Mode Exceptions**

#UD If the LOCK prefix is used.

**Real-Address Mode Exceptions**

Same exceptions as protected mode.

**Virtual-8086 Mode Exceptions**

Same exceptions as protected mode.

**Compatibility Mode Exceptions**

Same exceptions as protected mode.

**64-Bit Mode Exceptions**

#UD                      If in 64-bit mode.

**ADC—Add with Carry**

Opcode	Instruction	Op/En	64-bit Mode	Compat/Leg Mode	Description
14 <i>ib</i>	ADC AL, <i>imm8</i>	I	Valid	Valid	Add with carry <i>imm8</i> to AL.
15 <i>iw</i>	ADC AX, <i>imm16</i>	I	Valid	Valid	Add with carry <i>imm16</i> to AX.
15 <i>id</i>	ADC EAX, <i>imm32</i>	I	Valid	Valid	Add with carry <i>imm32</i> to EAX.
REX.W + 15 <i>id</i>	ADC RAX, <i>imm32</i>	I	Valid	N.E.	Add with carry <i>imm32</i> sign extended to 64-bits to RAX.
80 /2 <i>ib</i>	ADC <i>r/m8</i> , <i>imm8</i>	MI	Valid	Valid	Add with carry <i>imm8</i> to <i>r/m8</i> .
REX + 80 /2 <i>ib</i>	ADC <i>r/m8</i> <sup>*</sup> , <i>imm8</i>	MI	Valid	N.E.	Add with carry <i>imm8</i> to <i>r/m8</i> .
81 /2 <i>iw</i>	ADC <i>r/m16</i> , <i>imm16</i>	MI	Valid	Valid	Add with carry <i>imm16</i> to <i>r/m16</i> .
81 /2 <i>id</i>	ADC <i>r/m32</i> , <i>imm32</i>	MI	Valid	Valid	Add with CF <i>imm32</i> to <i>r/m32</i> .
REX.W + 81 /2 <i>id</i>	ADC <i>r/m64</i> , <i>imm32</i>	MI	Valid	N.E.	Add with CF <i>imm32</i> sign extended to 64-bits to <i>r/m64</i> .
83 /2 <i>ib</i>	ADC <i>r/m16</i> , <i>imm8</i>	MI	Valid	Valid	Add with CF sign-extended <i>imm8</i> to <i>r/m16</i> .
83 /2 <i>ib</i>	ADC <i>r/m32</i> , <i>imm8</i>	MI	Valid	Valid	Add with CF sign-extended <i>imm8</i> into <i>r/m32</i> .
REX.W + 83 /2 <i>ib</i>	ADC <i>r/m64</i> , <i>imm8</i>	MI	Valid	N.E.	Add with CF sign-extended <i>imm8</i> into <i>r/m64</i> .
10 /r	ADC <i>r/m8</i> , <i>r8</i>	MR	Valid	Valid	Add with carry byte register to <i>r/m8</i> .
REX + 10 /r	ADC <i>r/m8</i> <sup>*</sup> , <i>r8</i>	MR	Valid	N.E.	Add with carry byte register to <i>r/m64</i> .
11 /r	ADC <i>r/m16</i> , <i>r16</i>	MR	Valid	Valid	Add with carry <i>r16</i> to <i>r/m16</i> .
11 /r	ADC <i>r/m32</i> , <i>r32</i>	MR	Valid	Valid	Add with CF <i>r32</i> to <i>r/m32</i> .
REX.W + 11 /r	ADC <i>r/m64</i> , <i>r64</i>	MR	Valid	N.E.	Add with CF <i>r64</i> to <i>r/m64</i> .
12 /r	ADC <i>r8</i> , <i>r/m8</i>	RM	Valid	Valid	Add with carry <i>r/m8</i> to byte register.
REX + 12 /r	ADC <i>r8</i> <sup>*</sup> , <i>r/m8</i> <sup>*</sup>	RM	Valid	N.E.	Add with carry <i>r/m64</i> to byte register.
13 /r	ADC <i>r16</i> , <i>r/m16</i>	RM	Valid	Valid	Add with carry <i>r/m16</i> to <i>r16</i> .
13 /r	ADC <i>r32</i> , <i>r/m32</i>	RM	Valid	Valid	Add with CF <i>r/m32</i> to <i>r32</i> .
REX.W + 13 /r	ADC <i>r64</i> , <i>r/m64</i>	RM	Valid	N.E.	Add with CF <i>r/m64</i> to <i>r64</i> .

**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.

**Instruction Operand Encoding**

Op/En	Operand 1	Operand 2	Operand 3	Operand 4
RM	ModRM:reg ( <i>r</i> , <i>w</i> )	ModRM:r/m ( <i>r</i> )	NA	NA
MR	ModRM:r/m ( <i>r</i> , <i>w</i> )	ModRM:reg ( <i>r</i> )	NA	NA
MI	ModRM:r/m ( <i>r</i> , <i>w</i> )	<i>imm8</i>	NA	NA
I	AL/AX/EAX/RAX	<i>imm8</i>	NA	NA

**Description**

Adds the destination operand (first operand), the source operand (second operand), and the carry (CF) flag and stores the result in the destination operand. The destination operand can be a register or a memory location; the source operand can be an immediate, a register, or a memory location. (However, two memory operands cannot be used in one instruction.) The state of the CF flag represents a carry from a previous addition. When an immediate value is used as an operand, it is sign-extended to the length of the destination operand format.

The ADC instruction does not distinguish between signed or unsigned operands. Instead, the processor evaluates the result for both data types and sets the OF and CF flags to indicate a carry in the signed or unsigned result, respectively. The SF flag indicates the sign of the signed result.

The ADC instruction is usually executed as part of a multibyte or multiword addition in which an ADD instruction is followed by an ADC instruction.

This instruction can be used with a LOCK prefix to allow the instruction to be executed atomically.

In 64-bit mode, the instruction's default operation size is 32 bits. Using a REX prefix in the form of REX.R permits access to additional registers (R8-R15). Using a REX prefix in the form of REX.W promotes operation to 64 bits. See the summary chart at the beginning of this section for encoding data and limits.

## Operation

$DEST \leftarrow DEST + SRC + CF;$

## Intel C/C++ Compiler Intrinsic Equivalent

ADC: extern unsigned char \_addcarry\_u8(unsigned char c\_in, unsigned char src1, unsigned char src2, unsigned char \*sum\_out);

ADC: extern unsigned char \_addcarry\_u16(unsigned char c\_in, unsigned short src1, unsigned short src2, unsigned short \*sum\_out);

ADC: extern unsigned char \_addcarry\_u32(unsigned char c\_in, unsigned int src1, unsigned char int, unsigned int \*sum\_out);

ADC: extern unsigned char \_addcarry\_u64(unsigned char c\_in, unsigned \_\_int64 src1, unsigned \_\_int64 src2, unsigned \_\_int64 \*sum\_out);

## Flags Affected

The OF, SF, ZF, AF, CF, and PF flags are set according to the result.

## Protected Mode Exceptions

#GP(0)	If the destination is located in a non-writable segment. 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.
#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 but the destination is not a memory operand.

## 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.
#UD	If the LOCK prefix is used but the destination is not a memory operand.

## 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.
#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 but the destination is not a memory operand.

### 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.
#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 but the destination is not a memory operand.

## ADCX — Unsigned Integer Addition of Two Operands with Carry Flag

Opcode/ Instruction	Op/ En	64/32bit Mode Support	CPUID Feature Flag	Description
66 0F 38 F6 /r ADCX r32, r/m32	RM	V/V	ADX	Unsigned addition of r32 with CF, r/m32 to r32, writes CF.
66 REX.w 0F 38 F6 /r ADCX r64, r/m64	RM	V/NE	ADX	Unsigned addition of r64 with CF, r/m64 to r64, writes CF.

### Instruction Operand Encoding

Op/En	Operand 1	Operand 2	Operand 3	Operand 4
RM	ModRM:reg (r, w)	ModRM:r/m (r)	NA	NA

### Description

Performs an unsigned addition of the destination operand (first operand), the source operand (second operand) and the carry-flag (CF) and stores the result in the destination operand. The destination operand is a general-purpose register, whereas the source operand can be a general-purpose register or memory location. The state of CF can represent a carry from a previous addition. The instruction sets the CF flag with the carry generated by the unsigned addition of the operands.

The ADCX instruction is executed in the context of multi-precision addition, where we add a series of operands with a carry-chain. At the beginning of a chain of additions, we need to make sure the CF is in a desired initial state. Often, this initial state needs to be 0, which can be achieved with an instruction to zero the CF (e.g. XOR).

This instruction is supported in real mode and virtual-8086 mode. The operand size is always 32 bits if not in 64-bit mode.

In 64-bit mode, the default operation size is 32 bits. Using a REX Prefix in the form of REX.R permits access to additional registers (R8-15). Using REX Prefix in the form of REX.W promotes operation to 64 bits.

ADCX executes normally either inside or outside a transaction region.

Note: ADCX defines the OF flag differently than the ADD/ADC instructions as defined in *Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 2A*.

### Operation

IF OperandSize is 64-bit

THEN CF:DEST[63:0] ← DEST[63:0] + SRC[63:0] + CF;

ELSE CF:DEST[31:0] ← DEST[31:0] + SRC[31:0] + CF;

FI;

### Flags Affected

CF is updated based on result. OF, SF, ZF, AF and PF flags are unmodified.

### Intel C/C++ Compiler Intrinsic Equivalent

```
unsigned char __addcarryx_u32 (unsigned char c_in, unsigned int src1, unsigned int src2, unsigned int *sum_out);
```

```
unsigned char __addcarryx_u64 (unsigned char c_in, unsigned __int64 src1, unsigned __int64 src2, unsigned __int64 *sum_out);
```

### SIMD Floating-Point Exceptions

None

### Protected Mode Exceptions

#UD If the LOCK prefix is used.

If CPUID.(EAX=07H, ECX=0H):EBX.ADX[bit 19] = 0.

#SS(0) For an illegal address in the SS segment.

## INSTRUCTION SET REFERENCE, A-L

#GP(0)	For an illegal memory operand effective address in the CS, DS, ES, FS or GS segments. If the DS, ES, FS, or GS register is used to access memory and it contains a null segment selector.
#PF(fault-code)	For a page fault.
#AC(0)	If alignment checking is enabled and an unaligned memory reference is made while the current privilege level is 3.

### Real-Address Mode Exceptions

#UD	If the LOCK prefix is used. If CPUID.(EAX=07H, ECX=0H):EBX.ADX[bit 19] = 0.
#SS(0)	For an illegal address in the SS segment.
#GP(0)	If any part of the operand lies outside the effective address space from 0 to FFFFH.

### Virtual-8086 Mode Exceptions

#UD	If the LOCK prefix is used. If CPUID.(EAX=07H, ECX=0H):EBX.ADX[bit 19] = 0.
#SS(0)	For an illegal address in the SS segment.
#GP(0)	If any part of the operand lies outside the effective address space from 0 to FFFFH.
#PF(fault-code)	For a page fault.
#AC(0)	If alignment checking is enabled and an unaligned memory reference is made while the current privilege level is 3.

### Compatibility Mode Exceptions

Same exceptions as in protected mode.

### 64-Bit Mode Exceptions

#UD	If the LOCK prefix is used. If CPUID.(EAX=07H, ECX=0H):EBX.ADX[bit 19] = 0.
#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.
#PF(fault-code)	For a page fault.
#AC(0)	If alignment checking is enabled and an unaligned memory reference is made while the current privilege level is 3.

**ADD—Add**

Opcode	Instruction	Op/En	64-bit Mode	Compat/Leg Mode	Description
04 <i>ib</i>	ADD AL, <i>imm8</i>	I	Valid	Valid	Add <i>imm8</i> to AL.
05 <i>iw</i>	ADD AX, <i>imm16</i>	I	Valid	Valid	Add <i>imm16</i> to AX.
05 <i>id</i>	ADD EAX, <i>imm32</i>	I	Valid	Valid	Add <i>imm32</i> to EAX.
REX.W + 05 <i>id</i>	ADD RAX, <i>imm32</i>	I	Valid	N.E.	Add <i>imm32</i> sign-extended to 64-bits to RAX.
80 /0 <i>ib</i>	ADD r/m8, <i>imm8</i>	MI	Valid	Valid	Add <i>imm8</i> to r/m8.
REX + 80 /0 <i>ib</i>	ADD r/m8, <i>imm8</i>	MI	Valid	N.E.	Add sign-extended <i>imm8</i> to r/m64.
81 /0 <i>iw</i>	ADD r/m16, <i>imm16</i>	MI	Valid	Valid	Add <i>imm16</i> to r/m16.
81 /0 <i>id</i>	ADD r/m32, <i>imm32</i>	MI	Valid	Valid	Add <i>imm32</i> to r/m32.
REX.W + 81 /0 <i>id</i>	ADD r/m64, <i>imm32</i>	MI	Valid	N.E.	Add <i>imm32</i> sign-extended to 64-bits to r/m64.
83 /0 <i>ib</i>	ADD r/m16, <i>imm8</i>	MI	Valid	Valid	Add sign-extended <i>imm8</i> to r/m16.
83 /0 <i>ib</i>	ADD r/m32, <i>imm8</i>	MI	Valid	Valid	Add sign-extended <i>imm8</i> to r/m32.
REX.W + 83 /0 <i>ib</i>	ADD r/m64, <i>imm8</i>	MI	Valid	N.E.	Add sign-extended <i>imm8</i> to r/m64.
00 /r	ADD r/m8, r8	MR	Valid	Valid	Add r8 to r/m8.
REX + 00 /r	ADD r/m8, r8*	MR	Valid	N.E.	Add r8 to r/m8.
01 /r	ADD r/m16, r16	MR	Valid	Valid	Add r16 to r/m16.
01 /r	ADD r/m32, r32	MR	Valid	Valid	Add r32 to r/m32.
REX.W + 01 /r	ADD r/m64, r64	MR	Valid	N.E.	Add r64 to r/m64.
02 /r	ADD r8, r/m8	RM	Valid	Valid	Add r/m8 to r8.
REX + 02 /r	ADD r8, r/m8*	RM	Valid	N.E.	Add r/m8 to r8.
03 /r	ADD r16, r/m16	RM	Valid	Valid	Add r/m16 to r16.
03 /r	ADD r32, r/m32	RM	Valid	Valid	Add r/m32 to r32.
REX.W + 03 /r	ADD r64, r/m64	RM	Valid	N.E.	Add r/m64 to r64.

**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.

**Instruction Operand Encoding**

Op/En	Operand 1	Operand 2	Operand 3	Operand 4
RM	ModRM:reg (r, w)	ModRM:r/m (r)	NA	NA
MR	ModRM:r/m (r, w)	ModRM:reg (r)	NA	NA
MI	ModRM:r/m (r, w)	imm8	NA	NA
I	AL/AX/EAX/RAX	imm8	NA	NA

**Description**

Adds the destination operand (first operand) and the source operand (second operand) and then stores the result in the destination operand. The destination operand can be a register or a memory location; the source operand can be an immediate, a register, or a memory location. (However, two memory operands cannot be used in one instruction.) When an immediate value is used as an operand, it is sign-extended to the length of the destination operand format.

The ADD instruction performs integer addition. It evaluates the result for both signed and unsigned integer operands and sets the CF and OF flags to indicate a carry (overflow) in the signed or unsigned result, respectively. The SF flag indicates the sign of the signed result.

This instruction can be used with a LOCK prefix to allow the instruction to be executed atomically.

In 64-bit mode, the instruction's default operation size is 32 bits. Using a REX prefix in the form of REX.R permits access to additional registers (R8-R15). Using a REX prefix in the form of REX.W promotes operation to 64 bits. See the summary chart at the beginning of this section for encoding data and limits.

### Operation

DEST ← DEST + SRC;

### Flags Affected

The OF, SF, ZF, AF, CF, and PF flags are set according to the result.

### Protected Mode Exceptions

#GP(0)	If the destination is located in a non-writable segment. 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.
#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 but the destination is not a memory operand.

### 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.
#UD	If the LOCK prefix is used but the destination is not a memory operand.

### 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.
#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 but the destination is not a memory operand.

### 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.
#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 but the destination is not a memory operand.