x86 integer instructions

This is the full 8086/8088 instruction set. Most if not all of these instructions are available in 32-bit mode; they just operate on 32-bit registers (**eax**, **ebx**, etc.) and values instead of their 16-bit (**ax**, **bx**, etc.) counterparts. See also [x86 assembly language](https://en.wikipedia.org/wiki/X86_assembly_language) for a quick tutorial for this processor family. The updated instruction set is also grouped according to architecture ([i386](https://en.wikipedia.org/wiki/I386), [i486](https://en.wikipedia.org/wiki/I486), [i686](https://en.wikipedia.org/wiki/I686)) and more generally is referred to as [x86\_32](https://en.wikipedia.org/wiki/X86_32) and [x86\_64](https://en.wikipedia.org/wiki/X86_64) (also known as [AMD64](https://en.wikipedia.org/wiki/AMD64)).

**Original 8086/8088 instructions**

| **Original 8086/8088 instruction set** | | | |
| --- | --- | --- | --- |
| **Instruction** | **Meaning** | **Notes** | **Opcode** |
| [AAA](https://en.wikipedia.org/wiki/Intel_BCD_opcodes) | ASCII adjust AL after addition | used with unpacked [binary coded decimal](https://en.wikipedia.org/wiki/Binary_coded_decimal) | 0x37 |
| AAD | ASCII adjust AX before division | 8086/8088 datasheet documents only base 10 version of the AAD instruction ([opcode](https://en.wikipedia.org/wiki/Opcode" \o "Opcode) 0xD5 0x0A), but any other base will work. Later Intel's documentation has the generic form too. NEC V20 and V30 (and possibly other NEC V-series CPUs) always use base 10, and ignore the argument, causing a number of incompatibilities | 0xD5 |
| AAM | ASCII adjust AX after multiplication | Only base 10 version (Operand is 0xA) is documented, see notes for AAD | 0xD4 |
| AAS | ASCII adjust AL after subtraction |  | 0x3f |
| ADC | Add with carry | destination := destination + source + [carry\_flag](https://en.wikipedia.org/wiki/Carry_flag) | 0x10…0x15, 0x80/2…0x83/2 |
| ADD | Add | (1) r/m += r/imm; (2) r += m/imm; | 0x00…0x05, 0x80/0…0x83/0 |
| AND | [Logical AND](https://en.wikipedia.org/wiki/Logical_conjunction) | (1) r/m &= r/imm; (2) r &= m/imm; | 0x20…0x25, 0x80/4…0x83/4 |
| CALL | Call procedure | push eip*; eip points to the instruction directly after the call* | 0x9A, 0xE8, 0xFF/2, 0xFF/3 |
| CBW | Convert byte to word |  | 0x98 |
| CLC | Clear [carry flag](https://en.wikipedia.org/wiki/Carry_flag) | CF = 0; | 0xF8 |
| CLD | Clear [direction flag](https://en.wikipedia.org/wiki/Direction_flag) | DF = 0; | 0xFC |
| [CLI](https://en.wikipedia.org/wiki/CLI_(x86_instruction)) | Clear [interrupt flag](https://en.wikipedia.org/wiki/IF_(x86_flag)) | IF = 0; | 0xFA |
| CMC | Complement carry flag |  | 0xF5 |
| CMP | Compare operands |  | 0x38…0x3D, 0x80/7…0x83/7 |
| CMPSB | Compare bytes in memory |  | 0xA6 |
| CMPSW | Compare words |  | 0xA7 |
| CWD | Convert word to doubleword |  | 0x99 |
| [DAA](https://en.wikipedia.org/wiki/Intel_BCD_opcodes) | Decimal adjust AL after addition | (used with packed [binary coded decimal](https://en.wikipedia.org/wiki/Binary_coded_decimal)) | 0x27 |
| [DAS](https://en.wikipedia.org/wiki/Intel_BCD_opcodes) | Decimal adjust AL after subtraction |  | 0x2F |
| DEC | Decrement by 1 |  | 0x48, 0xFE/1, 0xFF/1 |
| DIV | Unsigned divide | DX:AX = DX:AX / r/m; resulting DX == remainder | 0xF6/6, 0xF7/6 |
| ESC | Used with [floating-point unit](https://en.wikipedia.org/wiki/Floating-point_unit) |  |  |
| [HLT](https://en.wikipedia.org/wiki/HLT_(x86_instruction)) | Enter halt state |  | 0xF4 |
| IDIV | Signed divide | DX:AX = DX:AX / r/m; resulting DX == remainder | 0xF6/7, 0xF7/7 |
| IMUL | Signed multiply | (1) DX:AX = AX \* r/m; (2) AX = AL \* r/m | 0x69, 0x6B, 0xF6/5, 0xF7/5, 0x0FAF |
| IN | Input from port | (1) AL = port[imm]; (2) AL = port[DX]; (3) AX = port[DX]; | 0xE4, 0xE5, 0xEC, 0xED |
| INC | Increment by 1 |  | 0x40, 0xFE/0, 0xFF/0 |
| [INT](https://en.wikipedia.org/wiki/INT_(x86_instruction)) | Call to [interrupt](https://en.wikipedia.org/wiki/Interrupt) |  | 0xCD |
| INTO | Call to interrupt if overflow |  | 0xCE |
| IRET | Return from interrupt |  | 0xCF |
| Jcc | [Jump if condition](https://en.wikipedia.org/wiki/Branch_(computer_science)) | (JA, JAE, JB, JBE, JC, JE, JG, JGE, JL, JLE, JNA, JNAE, JNB, JNBE, JNC, JNE, JNG, JNGE, JNL, JNLE, JNO, JNP, JNS, JNZ, JO, JP, JPE, JPO, JS, JZ) | 0x70…0x7F, 0xE3, 0x0F83, 0x0F87 |
| JCXZ | Jump if CX is zero |  | 0xE3 |
| [JMP](https://en.wikipedia.org/wiki/JMP_(x86_instruction)) | Jump |  | 0xE9…0xEB, 0xFF/4, 0xFF/5 |
| LAHF | Load FLAGS into AH register |  | 0x9F |
| LDS | Load pointer using DS |  | 0xC5 |
| LEA | [Load Effective Address](https://en.wikipedia.org/wiki/Load_Effective_Address) |  | 0x8D |
| LES | Load ES with pointer |  | 0xC4 |
| LOCK | Assert BUS LOCK# signal | (for multiprocessing) | 0xF0 |
| LODSB | Load string byte | **if** (DF==0) AL = \*SI++; **else** AL = \*SI--; | 0xAC |
| LODSW | Load string word | **if** (DF==0) AX = \*SI++; **else** AX = \*SI--; | 0xAD |
| LOOP/LOOPx | Loop control | (LOOPE, LOOPNE, LOOPNZ, LOOPZ) **if** (x && --CX) **goto** lbl; | 0xE0..0xE2 |
| MOV | Move | copies data from one location to another, (1) r/m = r; (2) r = r/m; |  |
| MOVSB | Move byte from string to string | **if** (DF==0)  \*(byte\*)DI++ = \*(byte\*)SI++;  **else**  \*(byte\*)DI-- = \*(byte\*)SI--; | 0xA4 |
| MOVSW | Move word from string to string | **if** (DF==0)  \*(word\*)DI++ = \*(word\*)SI++;  **else**  \*(word\*)DI-- = \*(word\*)SI--; | 0xA5 |
| MUL | Unsigned multiply | (1) DX:AX = AX \* r/m; (2) AX = AL \* r/m; |  |
| NEG | Two's complement negation | r/m \*= -1; |  |
| [NOP](https://en.wikipedia.org/wiki/NOP) | No operation | opcode equivalent to XCHG EAX, EAX | 0x90 |
| NOT | Negate the operand, [logical NOT](https://en.wikipedia.org/wiki/Bitwise_operation#NOT) | r/m ^= -1; |  |
| OR | [Logical OR](https://en.wikipedia.org/wiki/Logical_disjunction) | (1) r/m |= r/imm; (2) r |= m/imm; |  |
| OUT | Output to port | (1) port[imm] = AL; (2) port[DX] = AL; (3) port[DX] = AX; |  |
| POP | Pop data from [stack](https://en.wikipedia.org/wiki/Stack_(data_structure)) | r/m = \*SP++; POP CS (opcode 0x0F) works only on 8086/8088. Later CPUs use 0x0F as a prefix for newer instructions. |  |
| POPF | Pop [FLAGS register](https://en.wikipedia.org/wiki/FLAGS_register_(computing)) from stack | FLAGS = \*SP++; | 0x9D |
| PUSH | Push data onto stack | \*--SP = r/m; |  |
| PUSHF | Push FLAGS onto stack | \*--SP = FLAGS; | 0x9C |
| RCL | Rotate left (with carry) |  |  |
| RCR | Rotate right (with carry) |  |  |
| REPxx | Repeat MOVS/STOS/CMPS/LODS/SCAS | (REP, REPE, REPNE, REPNZ, REPZ) |  |
| RET | Return from procedure | Not a real instruction. The assembler will translate these to a RETN or a RETF depending on the memory model of the target system. |  |
| RETN | Return from near procedure |  |  |
| RETF | Return from far procedure |  |  |
| ROL | Rotate left |  |  |
| ROR | Rotate right |  |  |
| SAHF | Store AH into FLAGS |  | 0x9E |
| SAL | [Shift Arithmetically](https://en.wikipedia.org/wiki/Arithmetic_shift) left (signed shift left) | (1) r/m <<= 1; (2) r/m <<= CL; |  |
| SAR | Shift Arithmetically right (signed shift right) | (1) (signed) r/m >>= 1; (2) (signed) r/m >>= CL; |  |
| SBB | Subtraction with borrow | alternative 1-byte encoding of SBB AL, AL is available via [undocumented](https://en.wikipedia.org/wiki/X86_instruction_listings#Undocumented_instructions) SALC instruction |  |
| SCASB | Compare byte string |  | 0xAE |
| SCASW | Compare word string |  | 0xAF |
| SHL | [Shift](https://en.wikipedia.org/wiki/Logical_shift) left (unsigned shift left) |  |  |
| SHR | Shift right (unsigned shift right) |  |  |
| STC | Set carry flag | CF = 1; | 0xF9 |
| STD | Set direction flag | DF = 1; | 0xFD |
| [STI](https://en.wikipedia.org/wiki/STI_(x86_instruction)) | Set interrupt flag | IF = 1; | 0xFB |
| STOSB | Store byte in string | **if** (DF==0) \*ES:DI++ = AL; **else** \*ES:DI-- = AL; | 0xAA |
| STOSW | Store word in string | **if** (DF==0) \*ES:DI++ = AX; **else** \*ES:DI-- = AX; | 0xAB |
| SUB | Subtraction | (1) r/m -= r/imm; (2) r -= m/imm; |  |
| [TEST](https://en.wikipedia.org/wiki/TEST_(x86_instruction)) | Logical compare (AND) | (1) r/m & r/imm; (2) r & m/imm; |  |
| WAIT | Wait until not busy | Waits until BUSY# pin is inactive (used with [floating-point unit](https://en.wikipedia.org/wiki/Floating-point_unit)) | 0x9B |
| XCHG | Exchange data | r :=: r/m*;* A [spinlock](https://en.wikipedia.org/wiki/Spinlock) typically uses xchg as an [atomic operation](https://en.wikipedia.org/wiki/Atomic_operation). ([coma bug](https://en.wikipedia.org/wiki/Coma_bug)). |  |
| XLAT | Table look-up translation | behaves like MOV AL, [BX+AL] | 0xD7 |
| XOR | [Exclusive OR](https://en.wikipedia.org/wiki/Exclusive_or) | (1) r/m ^= r/imm; (2) r ^= m/imm; |  |

**Added in specific processors**[[edit](https://en.wikipedia.org/w/index.php?title=X86_instruction_listings&action=edit&section=3)]

**Added with**[**80186**](https://en.wikipedia.org/wiki/Intel_80186)**/**[**80188**](https://en.wikipedia.org/wiki/Intel_80188)[[edit](https://en.wikipedia.org/w/index.php?title=X86_instruction_listings&action=edit&section=4)]

|  |  |  |
| --- | --- | --- |
| **Instruction** | **Meaning** | **Notes** |
| BOUND | Check array index against bounds | raises software interrupt 5 if test fails |
| ENTER | Enter stack frame | Modifies stack for entry to procedure for high level language. Takes two operands: the amount of storage to be allocated on the stack and the nesting level of the procedure. |
| INS | Input from port to string | equivalent to  IN (E)AX, DX  MOV ES:[(E)DI], (E)AX  *; adjust (E)DI according to operand size and DF* |
| LEAVE | Leave stack frame | Releases the local stack storage created by the previous ENTER instruction. |
| OUTS | Output string to port | equivalent to  MOV (E)AX, DS:[(E)SI]  OUT DX, (E)AX  *; adjust (E)SI according to operand size and DF* |
| POPA | Pop all general purpose registers from stack | equivalent to  POP DI  POP SI  POP BP  POP AX *;no POP SP here, only ADD SP,2*  POP BX  POP DX  POP CX  POP AX |
| PUSHA | Push all general purpose registers onto stack | equivalent to  PUSH AX  PUSH CX  PUSH DX  PUSH BX  PUSH SP *; The value stored is the initial SP value*  PUSH BP  PUSH SI  PUSH DI |
| PUSH immediate | Push an immediate byte/word value onto the stack | equivalent to  PUSH 12h  PUSH 1200h |
| IMUL immediate | Signed multiplication of immediate byte/word value | equivalent to  IMUL BX,12h  IMUL DX,1200h  IMUL CX, DX, 12h  IMUL BX, SI, 1200h  IMUL DI, word ptr [BX+SI], 12h  IMUL SI, word ptr [BP-4], 1200h |
| SHL/SHR/SAL/SAR/ROL/ROR/RCL/RCR immediate | Rotate/shift bits with an immediate value greater than 1 | equivalent to  ROL AX,3  SHR BL,3 |

**Added with**[**80286**](https://en.wikipedia.org/wiki/80286)[[edit](https://en.wikipedia.org/w/index.php?title=X86_instruction_listings&action=edit&section=5)]

|  |  |  |
| --- | --- | --- |
| **Instruction** | **Meaning** | **Notes** |
| ARPL | Adjust RPL field of selector |  |
| CLTS | Clear task-switched flag in register CR0 |  |
| LAR | Load access rights byte |  |
| LGDT | Load global descriptor table |  |
| LIDT | Load interrupt descriptor table |  |
| LLDT | Load local descriptor table |  |
| LMSW | Load machine status word |  |
| [LOADALL](https://en.wikipedia.org/wiki/LOADALL) | Load all CPU registers, including internal ones such as GDT | Undocumented, 80286 and 80386 only |
| LSL | Load segment limit |  |
| [LTR](https://en.wikipedia.org/wiki/Load_Task_Register) | Load task register |  |
| SGDT | Store global descriptor table |  |
| SIDT | Store interrupt descriptor table |  |
| SLDT | Store local descriptor table |  |
| SMSW | Store machine status word |  |
| STR | Store task register |  |
| VERR | Verify a segment for reading |  |
| VERW | Verify a segment for writing |  |

**Added with**[**80386**](https://en.wikipedia.org/wiki/80386)[[edit](https://en.wikipedia.org/w/index.php?title=X86_instruction_listings&action=edit&section=6)]

|  |  |  |
| --- | --- | --- |
| **Instruction** | **Meaning** | **Notes** |
| [BSF](https://en.wikipedia.org/wiki/Find_first_set) | Bit scan forward |  |
| [BSR](https://en.wikipedia.org/wiki/Find_first_set) | Bit scan reverse |  |
| [BT](https://en.wikipedia.org/wiki/Bit_Test) | Bit test |  |
| [BTC](https://en.wikipedia.org/wiki/Bit_Test) | Bit test and complement |  |
| [BTR](https://en.wikipedia.org/wiki/Bit_Test) | Bit test and reset |  |
| [BTS](https://en.wikipedia.org/wiki/Bit_Test) | Bit test and set |  |
| CDQ | Convert double-word to quad-word | Sign-extends EAX into EDX, forming the quad-word EDX:EAX. Since (I)DIV uses EDX:EAX as its input, CDQ must be called after setting EAX if EDX is not manually initialized (as in 64/32 division) before (I)DIV. |
| CMPSD | Compare string double-word | Compares ES:[(E)DI] with DS:[SI] |
| CWDE | Convert word to double-word | Unlike CWD, CWDE sign-extends AX to EAX instead of AX to DX:AX |
| INSD | Input from port to string double-word |  |
| IRETx | Interrupt return; D suffix means 32-bit return, F suffix means do not generate epilogue code (i.e. LEAVE instruction) | Use IRETD rather than IRET in 32-bit situations |
| JECXZ | Jump if ECX is zero |  |
| LFS, LGS | Load far pointer |  |
| LSS | Load stack segment |  |
| LODSD | Load string double-word | can be prefixed with REP |
| LOOPW, LOOP*cc*W | Loop, conditional loop | Same as LOOP, LOOP*cc* for earlier processors |
| LOOPD, LOOPccD | Loop while equal | if (*cc* && --ECX) goto lbl;, *cc* = **Z**(ero), **E**(qual), **N**on**Z**ero, **N**(on)**E**(qual) |
| MOV to/from CR/DR/TR | Move to/from special registers | CR=control registers, DR=debug registers, TR=test registers (up to 80486) |
| MOVSD | Move string double-word | \*(dword\*)ES:EDI±± = (dword\*)ESI±±; (±± depends on DF) |
| MOVSX | Move with sign-extension | (long)r = (signed char) r/m; and similar |
| MOVZX | Move with zero-extension | (long)r = (unsigned char) r/m; and similar |
| OUTSD | Output to port from string double-word | port[DX] = \*(long\*)ESI±±; (±± depends on DF) |
| POPAD | Pop all double-word (32-bit) registers from stack | Does not pop register ESP off of stack |
| POPFD | Pop data into EFLAGS register |  |
| PUSHAD | Push all double-word (32-bit) registers onto stack |  |
| PUSHFD | Push EFLAGS register onto stack |  |
| SCASD | Scan string data double-word |  |
| SETcc | Set byte to one on condition, zero otherwise | (SETA, SETAE, SETB, SETBE, SETC, SETE, SETG, SETGE, SETL, SETLE, SETNA, SETNAE, SETNB, SETNBE, SETNC, SETNE, SETNG, SETNGE, SETNL, SETNLE, SETNO, SETNP, SETNS, SETNZ, SETO, SETP, SETPE, SETPO, SETS, SETZ) |
| SHLD | Shift left double-word |  |
| SHRD | Shift right double-word | r1 = r1>>CL ∣ r2<<(32-CL); Instead of CL, immediate 1 can be used |
| STOSD | Store string double-word | \*ES:EDI±± = EAX; (±± depends on DF, ES cannot be overridden) |

**Added with**[**80486**](https://en.wikipedia.org/wiki/80486)[[edit](https://en.wikipedia.org/w/index.php?title=X86_instruction_listings&action=edit&section=7)]

|  |  |  |
| --- | --- | --- |
| **Instruction** | **Meaning** | **Notes** |
| BSWAP | Byte Swap | r = r<<24 | r<<8&0x00FF0000 | r>>8&0x0000FF00 | r>>24; Only works for 32 bit registers |
| CMPXCHG | atomic CoMPare and eXCHanGe | See [Compare-and-swap](https://en.wikipedia.org/wiki/Compare-and-swap) / on later 80386 as undocumented opcode available |
| INVD | Invalidate Internal Caches | Flush internal caches |
| INVLPG | Invalidate [TLB](https://en.wikipedia.org/wiki/Translation_lookaside_buffer) Entry | Invalidate TLB Entry for page that contains data specified |
| WBINVD | Write Back and Invalidate Cache | Writes back all modified cache lines in the processor's internal cache to main memory and invalidates the internal caches. |
| XADD | eXchange and ADD | Exchanges the first operand with the second operand, then loads the sum of the two values into the destination operand. |

**Added with**[**Pentium**](https://en.wikipedia.org/wiki/Intel_P5)[[edit](https://en.wikipedia.org/w/index.php?title=X86_instruction_listings&action=edit&section=8)]

|  |  |  |
| --- | --- | --- |
| **Instruction** | **Meaning** | **Notes** |
| [CPUID](https://en.wikipedia.org/wiki/CPUID) | CPU IDentification | Returns data regarding processor identification and features, and returns data to the EAX, EBX, ECX, and EDX registers. Instruction functions specified by the EAX register.[[1]](https://en.wikipedia.org/wiki/X86_instruction_listings#cite_note-:0-1) This was also added to later [80486](https://en.wikipedia.org/wiki/80486) processors |
| CMPXCHG8B | CoMPare and eXCHanGe 8 bytes | Compare EDX:EAX with m64. If equal, set ZF and load ECX:EBX into m64. Else, clear ZF and load m64 into EDX:EAX. |
| [RDMSR](https://en.wikipedia.org/wiki/RDMSR) | ReaD from [Model-specific register](https://en.wikipedia.org/wiki/Model-specific_register) | Load [MSR](https://en.wikipedia.org/wiki/Model-specific_register) specified by ECX into EDX:EAX |
| [RDTSC](https://en.wikipedia.org/wiki/RDTSC) | ReaD Time Stamp Counter | Returns the number of processor ticks since the processor being "ONLINE" (since the last power on of system) |
| [WRMSR](https://en.wikipedia.org/wiki/WRMSR) | WRite to Model-Specific Register | Write the value in EDX:EAX to [MSR](https://en.wikipedia.org/wiki/Model-specific_register) specified by ECX |
| RSM [[1]](https://web.archive.org/web/20120312224625/http:/www.softeng.rl.ac.uk/st/archive/SoftEng/SESP/html/SoftwareTools/vtune/users_guide/mergedProjects/analyzer_ec/mergedProjects/reference_olh/mergedProjects/instructions/instruct32_hh/vc279.htm) | Resume from System Management Mode | This was introduced by the i[386SL](https://en.wikipedia.org/wiki/386SL#The_i386SL_variant) and later and is also in the i[486SL](https://en.wikipedia.org/wiki/Intel_80486SL) and later. Resumes from [System Management Mode](https://en.wikipedia.org/wiki/System_Management_Mode) (SMM) |

**Added with Pentium MMX**[[edit](https://en.wikipedia.org/w/index.php?title=X86_instruction_listings&action=edit&section=9)]

|  |  |  |
| --- | --- | --- |
| **Instruction** | **Meaning** | **Notes** |
| [RDPMC](https://en.wikipedia.org/w/index.php?title=RDPMC&action=edit&redlink=1) | Read the PMC [Performance Monitoring Counter] | Specified in the ECX register into registers EDX:EAX |

Also MMX registers and MMX support instructions were added. They are usable for both integer and floating point operations, see below.

**Added with**[**AMD K6**](https://en.wikipedia.org/wiki/AMD_K6)[[edit](https://en.wikipedia.org/w/index.php?title=X86_instruction_listings&action=edit&section=10)]

|  |  |  |
| --- | --- | --- |
| **Instruction** | **Meaning** | **Notes** |
| SYSCALL |  | functionally equivalent to SYSENTER |
| SYSRET |  | functionally equivalent to SYSEXIT |

AMD changed the CPUID detection bit for this feature from the K6-II on.

**Added with**[**Pentium Pro**](https://en.wikipedia.org/wiki/Pentium_Pro)[[edit](https://en.wikipedia.org/w/index.php?title=X86_instruction_listings&action=edit&section=11)]

|  |  |  |
| --- | --- | --- |
| **Instruction** | **Meaning** | **Notes** |
| CMOVcc | Conditional move | *(CMOVA, CMOVAE, CMOVB, CMOVBE, CMOVC, CMOVE, CMOVG, CMOVGE, CMOVL, CMOVLE, CMOVNA, CMOVNAE, CMOVNB, CMOVNBE, CMOVNC, CMOVNE, CMOVNG, CMOVNGE, CMOVNL, CMOVNLE, CMOVNO, CMOVNP, CMOVNS, CMOVNZ, CMOVO, CMOVP, CMOVPE, CMOVPO, CMOVS, CMOVZ)* |
| UD2 | Undefined Instruction | Generates an invalid opcode. This instruction is provided for software testing to explicitly generate an invalid opcode. The opcode for this instruction is reserved for this purpose. |

**Added with**[**Pentium II**](https://en.wikipedia.org/wiki/Pentium_II)[[edit](https://en.wikipedia.org/w/index.php?title=X86_instruction_listings&action=edit&section=12)]

|  |  |  |
| --- | --- | --- |
| **Instruction** | **Meaning** | **Notes** |
| SYSENTER | SYStem call ENTER | Sometimes called the Fast System Call instruction, this instruction was intended to increase the performance of operating system calls. Note that on the Pentium Pro, the [CPUID](https://en.wikipedia.org/wiki/CPUID) instruction incorrectly reports these instructions as available. |
| SYSEXIT | SYStem call EXIT |  |

**Added with**[**SSE**](https://en.wikipedia.org/wiki/Streaming_SIMD_Extensions)[[edit](https://en.wikipedia.org/w/index.php?title=X86_instruction_listings&action=edit&section=13)]

|  |  |  |  |
| --- | --- | --- | --- |
| **Instruction** | **Opcode** | **Meaning** | **Notes** |
| MASKMOVQ mm1, mm2 | 0F F7 /r | Masked Move of Quadword | Selectively write bytes from mm1 to memory location using the byte mask in mm2 |
| MOVNTPS m128, xmm1 | 0F 2B /r | Move Aligned Four Packed Single-FP Non Temporal | Move packed single-precision floating-point values from xmm1 to m128, minimizing pollution in the [cache hierarchy](https://en.wikipedia.org/wiki/Cache_hierarchy). |
| MOVNTQ m64, mm | 0F E7 /r | Move Quadword Using Non-Temporal Hint |  |
| NOP r/m16 | 0F 1F /0 | Multi-byte no-operation instruction. |  |
| NOP r/m32 |
| PREFETCHT0 | 0F 18 /1 | Prefetch Data from Address | Prefetch into all cache levels |
| PREFETCHT1 | 0F 18 /2 | Prefetch Data from Address | Prefetch into all cache levels EXCEPT[[2]](https://en.wikipedia.org/wiki/X86_instruction_listings#cite_note-2)[[3]](https://en.wikipedia.org/wiki/X86_instruction_listings#cite_note-3) L1 |
| PREFETCHT2 | 0F 18 /3 | Prefetch Data from Address | Prefetch into all cache levels EXCEPT L1 and L2 |
| PREFETCHNTA | 0F 18 /0 | Prefetch Data from Address | Prefetch to non-temporal cache structure, minimizing cache pollution. |
| SFENCE | 0F AE F8 | Store Fence | Processor hint to make sure all store operations that took place prior to the SFENCE call are globally visible |

**Added with**[**SSE2**](https://en.wikipedia.org/wiki/SSE2)[[edit](https://en.wikipedia.org/w/index.php?title=X86_instruction_listings&action=edit&section=14)]

|  |  |  |  |
| --- | --- | --- | --- |
| **Instruction** | **Opcode** | **Meaning** | **Notes** |
| CLFLUSH m8 | 0F AE /7 | Cache Line Flush | Invalidates the cache line that contains the linear address specified with the source operand from all levels of the processor cache hierarchy |
| LFENCE | 0F AE E8 | Load Fence | Serializes load operations. |
| MFENCE | 0F AE F0 | Memory Fence | Performs a serializing operation on all load and store instructions that were issued prior the MFENCE instruction. |
| MOVNTI m32, r32 | 0F C3 /r | Move Doubleword Non-Temporal | Move doubleword from r32 to m32, minimizing pollution in the cache hierarchy. |
| PAUSE | F3 90 | Spin Loop Hint | Provides a hint to the processor that the following code is a spin loop, for cacheability |

**Added with**[**SSE3**](https://en.wikipedia.org/wiki/SSE3)[[edit](https://en.wikipedia.org/w/index.php?title=X86_instruction_listings&action=edit&section=15)]

|  |  |  |
| --- | --- | --- |
| **Instruction** | **Meaning** | **Notes** |
| MONITOR EAX, ECX, EDX | Setup Monitor Address | Sets up a linear address range to be monitored by hardware and activates the monitor. |
| MWAIT EAX, ECX | Monitor Wait | Processor hint to stop instruction execution and enter an implementation-dependent optimized state until occurrence of a class of events. |

**Added with**[**SSE4.2**](https://en.wikipedia.org/wiki/SSE4.2)[[edit](https://en.wikipedia.org/w/index.php?title=X86_instruction_listings&action=edit&section=16)]

|  |  |  |  |
| --- | --- | --- | --- |
| **Instruction** | **Opcode** | **Meaning** | **Notes** |
| CRC32 r32, r/m8 | F2 0F 38 F0 /r | Accumulate CRC32 | Computes [CRC](https://en.wikipedia.org/wiki/Cyclic_redundancy_check) value using the CRC-32C (Castagnoli) polynomial 0x11EDC6F41 (normal form 0x1EDC6F41). This is the polynomial used in iSCSI. In contrast to the more popular one used in Ethernet, its parity is even, and it can thus detect any error with an odd number of changed bits. |
| CRC32 r32, r/m8 | F2 REX 0F 38 F0 /r |
| CRC32 r32, r/m16 | F2 0F 38 F1 /r |
| CRC32 r32, r/m32 | F2 0F 38 F1 /r |
| CRC32 r64, r/m8 | F2 REX.W 0F 38 F0 /r |
| CRC32 r64, r/m64 | F2 REX.W 0F 38 F1 /r |
| CRC32 r32, r/m8 | F2 0F 38 F0 /r |

**Added with**[**x86-64**](https://en.wikipedia.org/wiki/X86-64)[[edit](https://en.wikipedia.org/w/index.php?title=X86_instruction_listings&action=edit&section=17)]

|  |  |  |
| --- | --- | --- |
| **Instruction** | **Meaning** | **Notes** |
| CDQE | Sign extend EAX into RAX |  |
| CQO | Sign extend RAX into RDX:RAX |  |
| CMPSQ | CoMPare String Quadword |  |
| CMPXCHG16B | CoMPare and eXCHanGe 16 Bytes |  |
| IRETQ | 64-bit Return from Interrupt |  |
| JRCXZ | Jump if RCX is zero |  |
| LODSQ | LOaD String Quadword |  |
| MOVSXD | MOV with Sign Extend 32-bit to 64-bit |  |
| POPFQ | POP RFLAGS Register |  |
| PUSHFQ | PUSH RFLAGS Register |  |
| RDTSCP | ReaD Time Stamp Counter and Processor ID |  |
| SCASQ | SCAn String Quadword |  |
| STOSQ | STOre String Quadword |  |
| SWAPGS | Exchange GS base with KernelGSBase MSR |  |

**Added with**[**AMD-V**](https://en.wikipedia.org/wiki/X86_virtualization#AMD_virtualization_.28AMD-V.29)[[edit](https://en.wikipedia.org/w/index.php?title=X86_instruction_listings&action=edit&section=18)]

|  |  |  |
| --- | --- | --- |
| **Instruction** | **Meaning** | **Notes** |
| CLGI | Clear Global Interrupt Flag | Clears the GIF |
| INVLPGA | Invalidate TLB entry in a specified ASID | Invalidates the TLB mapping for the virtual page specified in RAX and the ASID specified in ECX. |
| MOV(CRn) | Move to or from control registers | Moves 32- or 64-bit contents to control register and vice versa. |
| SKINIT | Secure Init and Jump with Attestation | Verifiable startup of trusted software based on secure hash comparison |
| STGI | Set Global Interrupt Flag | Sets the GIF. |
| VMLOAD | Load state From VMCB | Loads a subset of processor state from the VMCB specified by the physical address in the RAX register. |
| VMMCALL | Call VMM | Used exclusively to communicate with VMM |
| VMRUN | Run virtual machine | Performs a switch to the guest OS. |
| VMSAVE | Save state To VMCB | Saves additional guest state to VMCB. |

**Added with**[**Intel VT-x**](https://en.wikipedia.org/wiki/Intel_VT-x)[[edit](https://en.wikipedia.org/w/index.php?title=X86_instruction_listings&action=edit&section=19)]

|  |  |  |  |
| --- | --- | --- | --- |
| **Instruction** | **Meaning** | **Notes** | **Opcode** |
| VMPTRLD | Load Pointer to Virtual-Machine Control Structure | Loads the current VMCS pointer from memory. | 0x0F 0xC7/6 |
| VMPTRST | Store Pointer to Virtual-Machine Control Structure | Stores the current-VMCS pointer into a specified memory address. The operand of this instruction is always 64 bits and is always in memory. | 0x0F 0xC7/7 |
| VMCLEAR | Clear Virtual-Machine Control Structure | Writes any cached data to the VMCS | 0x66 0x0F 0xC7/6 |
| VMREAD | Read Field from Virtual-Machine Control Structure | Reads out a field in the VMCS | 0x0F 0x78 |
| VMWRITE | Write Field to Virtual-Machine Control Structure | Modifies a field in the VMCS | 0x0F 0x79 |
| VMCALL | Call to VM Monitor | Calls VM Monitor function from Guest System | 0x0F 0x01 0xC1 |
| VMLAUNCH | Launch Virtual Machine | Launch virtual machine managed by current VMCS | 0x0F 0x01 0xC2 |
| VMRESUME | Resume Virtual Machine | Resume virtual machine managed by current VMCS | 0x0F 0x01 0xC3 |
| VMXOFF | Leave VMX Operation | Stops hardware supported virualisation environment | 0x0F 0x01 0xC4 |
| VMXON | Enter VMX Operation | Enters hardware supported virualisation environment | 0xF3 0x0F 0xC7/6 |

**Added with**[**ABM**](https://en.wikipedia.org/wiki/Advanced_Bit_Manipulation)[[edit](https://en.wikipedia.org/w/index.php?title=X86_instruction_listings&action=edit&section=20)]

[LZCNT](https://en.wikipedia.org/wiki/LZCNT), [POPCNT](https://en.wikipedia.org/wiki/Hamming_weight) (POPulation CouNT) - advanced bit manipulation

**Added with**[**BMI1**](https://en.wikipedia.org/wiki/Bit_Manipulation_Instruction_Sets#BMI1_.28Bit_Manipulation_Instruction_Set_1.29)[[edit](https://en.wikipedia.org/w/index.php?title=X86_instruction_listings&action=edit&section=21)]

ANDN, BEXTR, BLSI, BLSMSK, BLSR, TZCNT

**Added with**[**BMI2**](https://en.wikipedia.org/wiki/Bit_Manipulation_Instruction_Sets#BMI2_.28Bit_Manipulation_Instruction_Set_2.29)[[edit](https://en.wikipedia.org/w/index.php?title=X86_instruction_listings&action=edit&section=22)]

BZHI, MULX, PDEP, PEXT, RORX, SARX, SHRX, SHLX

**Added with**[**TBM**](https://en.wikipedia.org/wiki/Bit_Manipulation_Instruction_Sets#TBM_.28Trailing_Bit_Manipulation.29)[[edit](https://en.wikipedia.org/w/index.php?title=X86_instruction_listings&action=edit&section=23)]

BEXTR, BLCFILL, BLCI, BLCIC, BLCMASK, BLCS, BLSFILL, BLSIC, T1MSKC, TZMSK

[x87](https://en.wikipedia.org/wiki/X87) floating-point instructions[[edit](https://en.wikipedia.org/w/index.php?title=X86_instruction_listings&action=edit&section=24)]

**Original**[**8087**](https://en.wikipedia.org/wiki/8087)**instructions**[[edit](https://en.wikipedia.org/w/index.php?title=X86_instruction_listings&action=edit&section=25)]

|  |  |  |
| --- | --- | --- |
| **Instruction** | **Meaning** | **Notes** |
| F2XM1 | {\displaystyle 2^{x}-1} | more precise than {\displaystyle 2^{x}} for *x* close to zero |
| FABS | Absolute value |  |
| FADD | Add |  |
| FADDP | Add and pop |  |
| FBLD | Load BCD |  |
| FBSTP | Store BCD and pop |  |
| FCHS | Change sign |  |
| FCLEX | Clear exceptions |  |
| FCOM | Compare |  |
| FCOMP | Compare and pop |  |
| FCOMPP | Compare and pop twice |  |
| FDECSTP | Decrement floating point stack pointer |  |
| FDISI | Disable interrupts | 8087 only, otherwise FNOP |
| FDIV | Divide | [Pentium FDIV bug](https://en.wikipedia.org/wiki/Pentium_FDIV_bug) |
| FDIVP | Divide and pop |  |
| FDIVR | Divide reversed |  |
| FDIVRP | Divide reversed and pop |  |
| FENI | Enable interrupts | 8087 only, otherwise FNOP |
| FFREE | Free register |  |
| FIADD | Integer add |  |
| FICOM | Integer compare |  |
| FICOMP | Integer compare and pop |  |
| FIDIV | Integer divide |  |
| FIDIVR | Integer divide reversed |  |
| FILD | Load integer |  |
| FIMUL | Integer multiply |  |
| FINCSTP | Increment floating point stack pointer |  |
| FINIT | Initialize floating point processor |  |
| FIST | Store integer |  |
| FISTP | Store integer and pop |  |
| FISUB | Integer subtract |  |
| FISUBR | Integer subtract reversed |  |
| FLD | Floating point load |  |
| FLD1 | Load 1.0 onto stack |  |
| FLDCW | Load control word |  |
| FLDENV | Load environment state |  |
| FLDENVW | Load environment state, 16-bit |  |
| FLDL2E | Load log2(e) onto stack |  |
| FLDL2T | Load log2(10) onto stack |  |
| FLDLG2 | Load log10(2) onto stack |  |
| FLDLN2 | Load ln(2) onto stack |  |
| FLDPI | Load π onto stack |  |
| FLDZ | Load 0.0 onto stack |  |
| FMUL | Multiply |  |
| FMULP | Multiply and pop |  |
| FNCLEX | Clear exceptions, no wait |  |
| FNDISI | Disable interrupts, no wait | 8087 only, otherwise FNOP |
| FNENI | Enable interrupts, no wait | 8087 only, otherwise FNOP |
| FNINIT | Initialize floating point processor, no wait |  |
| FNOP | No operation |  |
| FNSAVE | Save FPU state, no wait, 8-bit |  |
| FNSAVEW | Save FPU state, no wait, 16-bit |  |
| FNSTCW | Store control word, no wait |  |
| FNSTENV | Store FPU environment, no wait |  |
| FNSTENVW | Store FPU environment, no wait, 16-bit |  |
| FNSTSW | Store status word, no wait |  |
| FPATAN | Partial arctangent |  |
| FPREM | Partial remainder |  |
| FPTAN | Partial tangent |  |
| FRNDINT | Round to integer |  |
| FRSTOR | Restore saved state |  |
| FRSTORW | Restore saved state | Perhaps not actually available in 8087 |
| FSAVE | Save FPU state |  |
| FSAVEW | Save FPU state, 16-bit |  |
| FSCALE | Scale by factor of 2 |  |
| FSQRT | Square root |  |
| FST | Floating point store |  |
| FSTCW | Store control word |  |
| FSTENV | Store FPU environment |  |
| FSTENVW | Store FPU environment, 16-bit |  |
| FSTP | Store and pop |  |
| FSTSW | Store status word |  |
| FSUB | Subtract |  |
| FSUBP | Subtract and pop |  |
| FSUBR | Reverse subtract |  |
| FSUBRP | Reverse subtract and pop |  |
| FTST | Test for zero |  |
| FWAIT | Wait while FPU is executing |  |
| FXAM | Examine condition flags |  |
| FXCH | Exchange registers |  |
| FXTRACT | Extract exponent and significand |  |
| FYL2X | *y* · log2 *x* | if *y* = log*b* 2, then the base-*b* logarithm is computed |
| FYL2XP1 | *y* · log2 (*x*+1) | more precise than log2 *z* if x is close to zero |

**Added in specific processors**[[edit](https://en.wikipedia.org/w/index.php?title=X86_instruction_listings&action=edit&section=26)]

**Added with**[**80287**](https://en.wikipedia.org/wiki/X87#80287)[[edit](https://en.wikipedia.org/w/index.php?title=X86_instruction_listings&action=edit&section=27)]

|  |  |  |
| --- | --- | --- |
| **Instruction** | **Meaning** | **Notes** |
| FSETPM | Set protected mode | 80287 only, otherwise FNOP |

**Added with**[**80387**](https://en.wikipedia.org/wiki/X87#80387)[[edit](https://en.wikipedia.org/w/index.php?title=X86_instruction_listings&action=edit&section=28)]

|  |  |  |
| --- | --- | --- |
| **Instruction** | **Meaning** | **Notes** |
| FCOS | Cosine |  |
| FLDENVD | Load environment state, 32-bit |  |
| FSAVED | Save FPU state, 32-bit |  |
| FSTENVD | Store FPU environment, 32-bit |  |
| FPREM1 | Partial remainder | Computes IEEE remainder |
| FRSTORD | Restore saved state, 32-bit |  |
| FSIN | Sine |  |
| FSINCOS | Sine and cosine |  |
| FSTENVD | Store FPU environment, 32-bit |  |
| FUCOM | Unordered compare |  |
| FUCOMP | Unordered compare and pop |  |
| FUCOMPP | Unordered compare and pop twice |  |

**Added with**[**Pentium Pro**](https://en.wikipedia.org/wiki/Pentium_Pro)[[edit](https://en.wikipedia.org/w/index.php?title=X86_instruction_listings&action=edit&section=29)]

* [FCMOV](https://en.wikipedia.org/wiki/FCMOV) variants: FCMOVB, FCMOVBE, FCMOVE, FCMOVNB, FCMOVNBE, FCMOVNE, FCMOVNU, FCMOVU
* [FCOMI](https://en.wikipedia.org/w/index.php?title=FCOMI&action=edit&redlink=1) variants: FCOMI, FCOMIP, FUCOMI, FUCOMIP

**Added with**[**SSE**](https://en.wikipedia.org/wiki/Streaming_SIMD_Extensions)[[edit](https://en.wikipedia.org/w/index.php?title=X86_instruction_listings&action=edit&section=30)]

FXRSTOR, FXSAVE

These are also supported on later Pentium IIs which do not contain SSE support

**Added with SSE3**[[edit](https://en.wikipedia.org/w/index.php?title=X86_instruction_listings&action=edit&section=31)]

FISTTP (x87 to integer conversion with truncation regardless of status word)

[SIMD](https://en.wikipedia.org/wiki/SIMD) instructions[[edit](https://en.wikipedia.org/w/index.php?title=X86_instruction_listings&action=edit&section=32)]

[**MMX**](https://en.wikipedia.org/wiki/MMX_(instruction_set))**instructions**[[edit](https://en.wikipedia.org/w/index.php?title=X86_instruction_listings&action=edit&section=33)]

MMX instructions operate on the mm registers, which are 64 bit wide.

**Original MMX instructions**[[edit](https://en.wikipedia.org/w/index.php?title=X86_instruction_listings&action=edit&section=34)]

*Added with*[*Pentium MMX*](https://en.wikipedia.org/wiki/Pentium_MMX)

|  |  |  |  |
| --- | --- | --- | --- |
| **Instruction** | **Opcode** | **Meaning** | **Notes** |
| EMMS | 0F 77 | Empty MMX Technology State | Marks all x87 FPU registers for use by FPU |
| MOVD mm, r/m32 | 0F 6E /r | Move doubleword |  |
| MOVD r/m32, mm | 0F 7E /r | Move doubleword |  |
| MOVQ mm/m64, mm | 0F 7F /r | Move quadword |  |
| MOVQ mm, mm/m64 | 0F 6F /r | Move quadword |  |
| MOVQ mm, r/m64 | REX.W + 0F 6E /r | Move quadword |  |
| MOVQ r/m64, mm | REX.W + 0F 7E /r | Move quadword |  |
| PACKSSDW mm1, mm2/m64 | 0F 6B /r | Pack doublewords to words (signed with saturation) |  |
| PACKSSWB mm1, mm2/m64 | 0F 63 /r | Pack words to bytes (signed with saturation) |  |
| PACKUSWB mm, mm/m64 | 0F 67 /r | Pack words to bytes (unsigned with saturation) |  |
| PADDB mm, mm/m64 | 0F FC /r | Add packed byte integers |  |
| PADDW mm, mm/m64 | 0F FD /r | Add packed word integers |  |
| PADDD mm, mm/m64 | 0F FE /r | Add packed doubleword integers |  |
| PADDQ mm, mm/m64 | 0F D4 /r | Add packed quadword integers |  |
| PADDSB mm, mm/m64 | 0F EC /r | Add packed signed byte integers and saturate |  |
| PADDSW mm, mm/m64 | 0F ED /r | Add packed signed word integers and saturate |  |
| PADDUSB mm, mm/m64 | 0F DC /r | Add packed unsigned byte integers and saturate |  |
| PADDUSW mm, mm/m64 | 0F DD /r | Add packed unsigned word integers and saturate |  |
| PAND mm, mm/m64 | 0F DB /r | Bitwise AND |  |
| PANDN mm, mm/m64 | 0F DF /r | Bitwise AND NOT |  |
| POR mm, mm/m64 | 0F EB /r | Bitwise OR |  |
| PXOR mm, mm/m64 | 0F EF /r | Bitwise XOR |  |
| PCMPEQB mm, mm/m64 | 0F 74 /r | Compare packed bytes for equality |  |
| PCMPEQW mm, mm/m64 | 0F 75 /r | Compare packed words for equality |  |
| PCMPEQD mm, mm/m64 | 0F 76 /r | Compare packed doublewords for equality |  |
| PCMPGTB mm, mm/m64 | 0F 64 /r | Compare packed signed byte integers for greater than |  |
| PCMPGTW mm, mm/m64 | 0F 65 /r | Compare packed signed word integers for greater than |  |
| PCMPGTD mm, mm/m64 | 0F 66 /r | Compare packed signed doubleword integers for greater than |  |
| PMADDWD mm, mm/m64 | 0F F5 /r | Multiply packed words, add adjacent doubleword results |  |
| PMULHW mm, mm/m64 | 0F E5 /r | Multiply packed signed word integers, store high 16 bits of results |  |
| PMULLW mm, mm/m64 | 0F D5 /r | Multiply packed signed word integers, store low 16 bits of results |  |
| PSLLW mm1, imm8 | 0F 71 /6 ib | Shift left words, shift in zeros |  |
| PSLLW mm, mm/m64 | 0F F1 /r | Shift left words, shift in zeros |  |
| PSLLD mm, imm8 | 0F 72 /6 ib | Shift left doublewords, shift in zeros |  |
| PSLLD mm, mm/m64 | 0F F2 /r | Shift left doublewords, shift in zeros |  |
| PSLLQ mm, imm8 | 0F 73 /6 ib | Shift left quadword, shift in zeros |  |
| PSLLQ mm, mm/m64 | 0F F3 /r | Shift left quadword, shift in zeros |  |
| PSRAD mm, imm8 | 0F 72 /4 ib | Shift right doublewords, shift in sign bits |  |
| PSRAD mm, mm/m64 | 0F E2 /r | Shift right doublewords, shift in sign bits |  |
| PSRAW mm, imm8 | 0F 71 /4 ib | Shift right words, shift in sign bits |  |
| PSRAW mm, mm/m64 | 0F E1 /r | Shift right words, shift in sign bits |  |
| PSRLW mm, imm8 | 0F 71 /2 ib | Shift right words, shift in zeros |  |
| PSRLW mm, mm/m64 | 0F D1 /r | Shift right words, shift in zeros |  |
| PSRLD mm, imm8 | 0F 72 /2 ib | Shift right doublewords, shift in zeros |  |
| PSRLD mm, mm/m64 | 0F D2 /r | Shift right doublewords, shift in zeros |  |
| PSRLQ mm, imm8 | 0F 73 /2 ib | Shift right quadword, shift in zeros |  |
| PSRLQ mm, mm/m64 | 0F D3 /r | Shift right quadword, shift in zeros |  |
| PSUBB mm, mm/m64 | 0F F8 /r | Subtract packed byte integers |  |
| PSUBW mm, mm/m64 | 0F F9 /r | Subtract packed word integers |  |
| PSUBD mm, mm/m64 | 0F FA /r | Subtract packed doubleword integers |  |
| PSUBSB mm, mm/m64 | 0F E8 /r | Subtract signed packed bytes with saturation |  |
| PSUBSW mm, mm/m64 | 0F E9 /r | Subtract signed packed words with saturation |  |
| PSUBUSB mm, mm/m64 | 0F D8 /r | Subtract unsigned packed bytes with saturation |  |
| PSUBUSW mm, mm/m64 | 0F D9 /r | Subtract unsigned packed words with saturation |  |
| PUNPCKHBW mm, mm/m64 | 0F 68 /r | Unpack and interleave high-order bytes |  |
| PUNPCKHWD mm, mm/m64 | 0F 69 /r | Unpack and interleave high-order words |  |
| PUNPCKHDQ mm, mm/m64 | 0F 6A /r | Unpack and interleave high-order doublewords |  |
| PUNPCKLBW mm, mm/m32 | 0F 60 /r | Unpack and interleave low-order bytes |  |
| PUNPCKLWD mm, mm/m32 | 0F 61 /r | Unpack and interleave low-order words |  |
| PUNPCKLDQ mm, mm/m32 | 0F 62 /r | Unpack and interleave low-order doublewords |  |

**MMX instructions added in specific processors**[[edit](https://en.wikipedia.org/w/index.php?title=X86_instruction_listings&action=edit&section=35)]

[**EMMI**](https://en.wikipedia.org/wiki/Extended_MMX)**instructions**[[edit](https://en.wikipedia.org/w/index.php?title=X86_instruction_listings&action=edit&section=36)]

*Added with*[*6x86MX*](https://en.wikipedia.org/wiki/6x86MX)*from*[*Cyrix*](https://en.wikipedia.org/wiki/Cyrix)*, deprecated now*

PAVEB, PADDSIW, PMAGW, PDISTIB, PSUBSIW, PMVZB, PMULHRW, PMVNZB, PMVLZB, PMVGEZB, PMULHRIW, PMACHRIW

**MMX instructions added with**[**MMX+**](https://en.wikipedia.org/wiki/Extended_MMX)**and SSE**[[edit](https://en.wikipedia.org/w/index.php?title=X86_instruction_listings&action=edit&section=37)]

The following MMX instruction were added with SSE. They are also available on the [Athlon](https://en.wikipedia.org/wiki/Athlon" \o "Athlon) under the name MMX+.

|  |  |  |
| --- | --- | --- |
| **Instruction** | **Opcode** | **Meaning** |
| PSHUFW mm1, mm2/m64, imm8 | 0F 70 /r ib | Shuffle Packed Words |
| PINSRW mm, r32/m16, imm8 | 0F C4 /r | Insert Word |
| PEXTRW reg, mm, imm8 | 0F C5 /r | Extract Word |
| PMOVMSKB reg, mm | 0F D7 /r | Move Byte Mask |
| PMINUB mm1, mm2/m64 | 0F DA /r | Minimum of Packed Unsigned Byte Integers |
| PMAXUB mm1, mm2/m64 | 0F DE /r | Maximum of Packed Unsigned Byte Integers |
| PAVGB mm1, mm2/m64 | 0F E0 /r | Average Packed Integers |
| PAVGW mm1, mm2/m64 | 0F E3 /r | Average Packed Integers |
| PMULHUW mm1, mm2/m64 | 0F E4 /r | Multiply Packed Unsigned Integers and Store High Result |
| PMINSW mm1, mm2/m64 | 0F EA /r | Minimum of Packed Signed Word Integers |
| PMAXSW mm1, mm2/m64 | 0F EE /r | Maximum of Packed Signed Word Integers |
| PSADBW mm1, mm2/m64 | 0F F6 /r | Compute Sum of Absolute Differences |

**MMX instructions added with SSE2**[[edit](https://en.wikipedia.org/w/index.php?title=X86_instruction_listings&action=edit&section=38)]

The following MMX instructions were added with SSE2:

|  |  |  |
| --- | --- | --- |
| **Instruction** | **Opcode** | **Meaning** |
| PSUBQ mm1, mm2/m64 | 0F FB /r | Subtract quadword integer |
| PMULUDQ mm1, mm2/m64 | 0F F4 /r | Multiply unsigned doubleword integer |

**MMX instructions added with SSSE3**[[edit](https://en.wikipedia.org/w/index.php?title=X86_instruction_listings&action=edit&section=39)]

|  |  |  |
| --- | --- | --- |
| **Instruction** | **Opcode** | **Meaning** |
| PSIGNB mm1, mm2/m64 | 0F 38 08 /r | Negate/zero/preserve packed byte integers depending on corresponding sign |
| PSIGNW mm1, mm2/m64 | 0F 38 09 /r | Negate/zero/preserve packed word integers depending on corresponding sign |
| PSIGND mm1, mm2/m64 | 0F 38 0A /r | Negate/zero/preserve packed doubleword integers depending on corresponding sign |
| PSHUFB mm1, mm2/m64 | 0F 38 00 /r | Shuffle bytes |
| PMULHRSW mm1, mm2/m64 | 0F 38 0B /r | Multiply 16-bit signed words, scale and round signed doublewords, pack high 16 bits |
| PMADDUBSW mm1, mm2/m64 | 0F 38 04 /r | Multiply signed and unsigned bytes, add horizontal pair of signed words, pack saturated signed-words |
| PHSUBW mm1, mm2/m64 | 0F 38 05 /r | Subtract and pack 16-bit signed integers horizontally |
| PHSUBSW mm1, mm2/m64 | 0F 38 07 /r | Subtract and pack 16-bit signed integer horizontally with saturation |
| PHSUBD mm1, mm2/m64 | 0F 38 06 /r | Subtract and pack 32-bit signed integers horizontally |
| PHADDSW mm1, mm2/m64 | 0F 38 03 /r | Add and pack 16-bit signed integers horizontally, pack saturated integers to mm1. |
| PHADDW mm1, mm2/m64 | 0F 38 01 /r | Add and pack 16-bit integers horizontally |
| PHADDD mm1, mm2/m64 | 0F 38 02 /r | Add and pack 32-bit integers horizontally |
| PALIGNR mm1, mm2/m64, imm8 | 0F 3A 0F /r ib | Concatenate destination and source operands, extract byte-aligned result shifted to the right |
| PABSB mm1, mm2/m64 | 0F 38 1C /r | Compute the absolute value of bytes and store unsigned result |
| PABSW mm1, mm2/m64 | 0F 38 1D /r | Compute the absolute value of 16-bit integers and store unsigned result |
| PABSD mm1, mm2/m64 | 0F 38 1E /r | Compute the absolute value of 32-bit integers and store unsigned result |

[**3DNow!**](https://en.wikipedia.org/wiki/3DNow!)**instructions**[[edit](https://en.wikipedia.org/w/index.php?title=X86_instruction_listings&action=edit&section=40)]

*Added with*[*K6-2*](https://en.wikipedia.org/wiki/K6-2)

FEMMS, PAVGUSB, PF2ID, PFACC, PFADD, PFCMPEQ, PFCMPGE, PFCMPGT, PFMAX, PFMIN, PFMUL, PFRCP, PFRCPIT1, PFRCPIT2, PFRSQIT1, PFRSQRT, PFSUB, PFSUBR, PI2FD, PMULHRW, PREFETCH, PREFETCHW

**3DNow!+ instructions**[[edit](https://en.wikipedia.org/w/index.php?title=X86_instruction_listings&action=edit&section=41)]

**Added with [Athlon](https://en.wikipedia.org/wiki/Athlon" \o "Athlon) and**[**K6-2+**](https://en.wikipedia.org/wiki/AMD_K6-III#K6-2.2B_.28180_nm.2C_mobile.29)[[edit](https://en.wikipedia.org/w/index.php?title=X86_instruction_listings&action=edit&section=42)]

PF2IW, PFNACC, PFPNACC, PI2FW, PSWAPD

**Added with**[**Geode GX**](https://en.wikipedia.org/wiki/Geode_GX)[[edit](https://en.wikipedia.org/w/index.php?title=X86_instruction_listings&action=edit&section=43)]

PFRSQRTV, PFRCPV

[**SSE**](https://en.wikipedia.org/wiki/Streaming_SIMD_Extensions)**instructions**[[edit](https://en.wikipedia.org/w/index.php?title=X86_instruction_listings&action=edit&section=44)]

*Added with*[*Pentium III*](https://en.wikipedia.org/wiki/Pentium_III)

SSE instructions operate on xmm registers, which are 128 bit wide.

SSE consists of the following SSE SIMD floating-point instructions:

|  |  |  |
| --- | --- | --- |
| **Instruction** | **Opcode** | **Meaning** |
| ANDPS\* xmm1, xmm2/m128 | 0F 54 /r | Bitwise Logical AND of Packed Single-Precision Floating-Point Values |
| ANDNPS\* xmm1, xmm2/m128 | 0F 55 /r | Bitwise Logical AND NOT of Packed Single-Precision Floating-Point Values |
| ORPS\* xmm1, xmm2/m128 | 0F 56 /r | Bitwise Logical OR of Single-Precision Floating-Point Values |
| XORPS\* xmm1, xmm2/m128 | 0F 57 /r | Bitwise Logical XOR for Single-Precision Floating-Point Values |
| MOVUPS xmm1, xmm2/m128 | 0F 10 /r | Move Unaligned Packed Single-Precision Floating-Point Values |
| MOVSS xmm1, xmm2/m32 | F3 0F 10 /r | Move Scalar Single-Precision Floating-Point Values |
| MOVUPS xmm2/m128, xmm1 | 0F 11 /r | Move Unaligned Packed Single-Precision Floating-Point Values |
| MOVSS xmm2/m32, xmm1 | F3 0F 11 /r | Move Scalar Single-Precision Floating-Point Values |
| MOVLPS xmm, m64 | 0F 12 /r | Move Low Packed Single-Precision Floating-Point Values |
| MOVHLPS xmm1, xmm2 | 0F 12 /r | Move Packed Single-Precision Floating-Point Values High to Low |
| MOVLPS m64, xmm | 0F 13 /r | Move Low Packed Single-Precision Floating-Point Values |
| UNPCKLPS xmm1, xmm2/m128 | 0F 14 /r | Unpack and Interleave Low Packed Single-Precision Floating-Point Values |
| UNPCKHPS xmm1, xmm2/m128 | 0F 15 /r | Unpack and Interleave High Packed Single-Precision Floating-Point Values |
| MOVHPS xmm, m64 | 0F 16 /r | Move High Packed Single-Precision Floating-Point Values |
| MOVLHPS xmm1, xmm2 | 0F 16 /r | Move Packed Single-Precision Floating-Point Values Low to High |
| MOVHPS m64, xmm | 0F 17 /r | Move High Packed Single-Precision Floating-Point Values |
| MOVAPS xmm1, xmm2/m128 | 0F 28 /r | Move Aligned Packed Single-Precision Floating-Point Values |
| MOVAPS xmm2/m128, xmm1 | 0F 29 /r | Move Aligned Packed Single-Precision Floating-Point Values |
| MOVMSKPS reg, xmm | 0F 50 /r | Extract Packed Single-Precision Floating-Point 4-bit Sign Mask. The upper bits of the register are filled with zeros. |
| CVTPI2PS xmm, mm/m64 | 0F 2A /r | Convert Packed Dword Integers to Packed Single-Precision FP Values |
| CVTSI2SS xmm, r/m32 | F3 0F 2A /r | Convert Dword Integer to Scalar Single-Precision FP Value |
| CVTSI2SS xmm, r/m64 | F3 REX.W 0F 2A /r | Convert Qword Integer to Scalar Single-Precision FP Value |
| MOVNTPS m128, xmm | 0F 2B /r | Store Packed Single-Precision Floating-Point Values Using Non-Temporal Hint |
| CVTTPS2PI mm, xmm/m64 | 0F 2C /r | Convert with Truncation Packed Single-Precision FP Values to Packed Dword Integers |
| CVTTSS2SI r32, xmm/m32 | F3 0F 2C /r | Convert with Truncation Scalar Single-Precision FP Value to Dword Integer |
| CVTTSS2SI r64, xmm1/m32 | F3 REX.W 0F 2C /r | Convert with Truncation Scalar Single-Precision FP Value to Qword Integer |
| CVTPS2PI mm, xmm/m64 | 0F 2D /r | Convert Packed Single-Precision FP Values to Packed Dword Integers |
| CVTSS2SI r32, xmm/m32 | F3 0F 2D /r | Convert Scalar Single-Precision FP Value to Dword Integer |
| CVTSS2SI r64, xmm1/m32 | F3 REX.W 0F 2D /r | Convert Scalar Single-Precision FP Value to Qword Integer |
| UCOMISS xmm1, xmm2/m32 | 0F 2E /r | Unordered Compare Scalar Single-Precision Floating-Point Values and Set EFLAGS |
| COMISS xmm1, xmm2/m32 | 0F 2F /r | Compare Scalar Ordered Single-Precision Floating-Point Values and Set EFLAGS |
| SQRTPS xmm1, xmm2/m128 | 0F 51 /r | Compute Square Roots of Packed Single-Precision Floating-Point Values |
| SQRTSS xmm1, xmm2/m32 | F3 0F 51 /r | Compute Square Root of Scalar Single-Precision Floating-Point Value |
| RSQRTPS xmm1, xmm2/m128 | 0F 52 /r | Compute Reciprocal of Square Root of Packed Single-Precision Floating-Point Value |
| RSQRTSS xmm1, xmm2/m32 | F3 0F 52 /r | Compute Reciprocal of Square Root of Scalar Single-Precision Floating-Point Value |
| RCPPS xmm1, xmm2/m128 | 0F 53 /r | Compute Reciprocal of Packed Single-Precision Floating-Point Values |
| RCPSS xmm1, xmm2/m32 | F3 0F 53 /r | Compute Reciprocal of Scalar Single-Precision Floating-Point Values |
| ADDPS xmm1, xmm2/m128 | 0F 58 /r | Add Packed Single-Precision Floating-Point Values |
| ADDSS xmm1, xmm2/m32 | F3 0F 58 /r | Add Scalar Single-Precision Floating-Point Values |
| MULPS xmm1, xmm2/m128 | 0F 59 /r | Multiply Packed Single-Precision Floating-Point Values |
| MULSS xmm1, xmm2/m32 | F3 0F 59 /r | Multiply Scalar Single-Precision Floating-Point Values |
| SUBPS xmm1, xmm2/m128 | 0F 5C /r | Subtract Packed Single-Precision Floating-Point Values |
| SUBSS xmm1, xmm2/m32 | F3 0F 5C /r | Subtract Scalar Single-Precision Floating-Point Values |
| MINPS xmm1, xmm2/m128 | 0F 5D /r | Return Minimum Packed Single-Precision Floating-Point Values |
| MINSS xmm1, xmm2/m32 | F3 0F 5D /r | Return Minimum Scalar Single-Precision Floating-Point Values |
| DIVPS xmm1, xmm2/m128 | 0F 5E /r | Divide Packed Single-Precision Floating-Point Values |
| DIVSS xmm1, xmm2/m32 | F3 0F 5E /r | Divide Scalar Single-Precision Floating-Point Values |
| MAXPS xmm1, xmm2/m128 | 0F 5F /r | Return Maximum Packed Single-Precision Floating-Point Values |
| MAXSS xmm1, xmm2/m32 | F3 0F 5F /r | Return Maximum Scalar Single-Precision Floating-Point Values |
| LDMXCSR m32 | 0F AE /2 | Load MXCSR Register State |
| STMXCSR m32 | 0F AE /3 | Store MXCSR Register State |
| CMPPS xmm1, xmm2/m128, imm8 | 0F C2 /r ib | Compare Packed Single-Precision Floating-Point Values |
| CMPSS xmm1, xmm2/m32, imm8 | F3 0F C2 /r ib | Compare Scalar Single-Precision Floating-Point Values |
| SHUFPS xmm1, xmm2/m128, imm8 | 0F C6 /r ib | Shuffle Packed Single-Precision Floating-Point Values |

* The floating point single bitwise operations ANDPS, ANDNPS, ORPS and XORPS produce the same result as the SSE2 integer (PAND, PANDN, POR, PXOR) and double ones (ANDPD, ANDNPD, ORPD, XORPD), but can introduce extra latency for domain changes when applied values of the wrong type.[[4]](https://en.wikipedia.org/wiki/X86_instruction_listings#cite_note-4)

[**SSE2**](https://en.wikipedia.org/wiki/SSE2)**instructions**[[edit](https://en.wikipedia.org/w/index.php?title=X86_instruction_listings&action=edit&section=45)]

*Added with*[*Pentium 4*](https://en.wikipedia.org/wiki/Pentium_4)

**SSE2 SIMD floating-point instructions**[[edit](https://en.wikipedia.org/w/index.php?title=X86_instruction_listings&action=edit&section=46)]

**SSE2 data movement instructions**[[edit](https://en.wikipedia.org/w/index.php?title=X86_instruction_listings&action=edit&section=47)]

|  |  |  |
| --- | --- | --- |
| **Instruction** | **Opcode** | **Meaning** |
| [MOVAPD](https://en.wikipedia.org/wiki/MOVAPD) xmm1, xmm2/m128 | 66 0F 28 /r | Move Aligned Packed Double-Precision Floating-Point Values |
| MOVAPD xmm2/m128, xmm1 | 66 0F 29 /r | Move Aligned Packed Double-Precision Floating-Point Values |
| MOVNTPD m128, xmm1 | 66 0F 2B /r | Store Packed Double-Precision Floating-Point Values Using Non-Temporal Hint |
| [MOVHPD](https://en.wikipedia.org/wiki/MOVHPD) xmm1, m64 | 66 0F 16 /r | Move High Packed Double-Precision Floating-Point Value |
| MOVHPD m64, xmm1 | 66 0F 17 /r | Move High Packed Double-Precision Floating-Point Value |
| MOVLPD xmm1, m64 | 66 0F 12 /r | Move Low Packed Double-Precision Floating-Point Value |
| MOVLPD m64, xmm1 | 66 0F 13/r | Move Low Packed Double-Precision Floating-Point Value |
| MOVUPD xmm1, xmm2/m128 | 66 0F 10 /r | Move Unaligned Packed Double-Precision Floating-Point Values |
| MOVUPD xmm2/m128, xmm1 | 66 0F 11 /r | Move Unaligned Packed Double-Precision Floating-Point Values |
| MOVMSKPD reg, xmm | 66 0F 50 /r | Extract Packed Double-Precision Floating-Point Sign Mask |
| MOVSD\* xmm1, xmm2/m64 | F2 0F 10 /r | Move or Merge Scalar Double-Precision Floating-Point Value |
| MOVSD xmm1/m64, xmm2 | F2 0F 11 /r | Move or Merge Scalar Double-Precision Floating-Point Value |

**SSE2 packed arithmetic instructions**[[edit](https://en.wikipedia.org/w/index.php?title=X86_instruction_listings&action=edit&section=48)]

|  |  |  |
| --- | --- | --- |
| **Instruction** | **Opcode** | **Meaning** |
| ADDPD xmm1, xmm2/m128 | 66 0F 58 /r | Add Packed Double-Precision Floating-Point Values |
| ADDSD xmm1, xmm2/m64 | F2 0F 58 /r | Add Low Double-Precision Floating-Point Value |
| DIVPD xmm1, xmm2/m128 | 66 0F 5E /r | Divide Packed Double-Precision Floating-Point Values |
| DIVSD xmm1, xmm2/m64 | F2 0F 5E /r | Divide Scalar Double-Precision Floating-Point Value |
| MAXPD xmm1, xmm2/m128 | 66 0F 5F /r | Maximum of Packed Double-Precision Floating-Point Values |
| MAXSD xmm1, xmm2/m64 | F2 0F 5F /r | Return Maximum Scalar Double-Precision Floating-Point Value |
| MINPD xmm1, xmm2/m128 | 66 0F 5D /r | Minimum of Packed Double-Precision Floating-Point Values |
| MINSD xmm1, xmm2/m64 | F2 0F 5D /r | Return Minimum Scalar Double-Precision Floating-Point Value |
| MULPD xmm1, xmm2/m128 | 66 0F 59 /r | Multiply Packed Double-Precision Floating-Point Values |
| MULSD xmm1,xmm2/m64 | F2 0F 59 /r | Multiply Scalar Double-Precision Floating-Point Value |
| SQRTPD xmm1, xmm2/m128 | 66 0F 51 /r | Square Root of Double-Precision Floating-Point Values |
| SQRTSD xmm1,xmm2/m64 | F2 0F 51/r | Compute Square Root of Scalar Double-Precision Floating-Point Value |
| SUBPD xmm1, xmm2/m128 | 66 0F 5C /r | Subtract Packed Double-Precision Floating-Point Values |
| SUBSD xmm1, xmm2/m64 | F2 0F 5C /r | Subtract Scalar Double-Precision Floating-Point Value |

**SSE2 logical instructions**[[edit](https://en.wikipedia.org/w/index.php?title=X86_instruction_listings&action=edit&section=49)]

|  |  |  |
| --- | --- | --- |
| **Instruction** | **Opcode** | **Meaning** |
| ANDPD xmm1, xmm2/m128 | 66 0F 54 /r | Bitwise Logical AND of Packed Double Precision Floating-Point Values |
| ANDNPD xmm1, xmm2/m128 | 66 0F 55 /r | Bitwise Logical AND NOT of Packed Double Precision Floating-Point Values |
| ORPD xmm1, xmm2/m128 | 66 0F 56/r | Bitwise Logical OR of Packed Double Precision Floating-Point Values |
| XORPD xmm1, xmm2/m128 | 66 0F 57/r | Bitwise Logical XOR of Packed Double Precision Floating-Point Values |

**SSE2 compare instructions**[[edit](https://en.wikipedia.org/w/index.php?title=X86_instruction_listings&action=edit&section=50)]

|  |  |  |
| --- | --- | --- |
| **Instruction** | **Opcode** | **Meaning** |
| CMPPD xmm1, xmm2/m128, imm8 | 66 0F C2 /r ib | Compare Packed Double-Precision Floating-Point Values |
| CMPSD\* xmm1, xmm2/m64, imm8 | F2 0F C2 /r ib | Compare Low Double-Precision Floating-Point Values |
| COMISD xmm1, xmm2/m64 | 66 0F 2F /r | Compare Scalar Ordered Double-Precision Floating-Point Values and Set EFLAGS |
| UCOMISD xmm1, xmm2/m64 | 66 0F 2E /r | Unordered Compare Scalar Double-Precision Floating-Point Values and Set EFLAGS |

**SSE2 shuffle and unpack instructions**[[edit](https://en.wikipedia.org/w/index.php?title=X86_instruction_listings&action=edit&section=51)]

|  |  |  |
| --- | --- | --- |
| **Instruction** | **Opcode** | **Meaning** |
| SHUFPD xmm1, xmm2/m128, imm8 | 66 0F C6 /r ib | Packed Interleave Shuffle of Pairs of Double-Precision Floating-Point Values |
| UNPCKHPD xmm1, xmm2/m128 | 66 0F 15 /r | Unpack and Interleave High Packed Double-Precision Floating-Point Values |
| UNPCKLPD xmm1, xmm2/m128 | 66 0F 14 /r | Unpack and Interleave Low Packed Double-Precision Floating-Point Values |

**SSE2 conversion instructions**[[edit](https://en.wikipedia.org/w/index.php?title=X86_instruction_listings&action=edit&section=52)]

|  |  |  |
| --- | --- | --- |
| **Instruction** | **Opcode** | **Meaning** |
| CVTDQ2PD xmm1, xmm2/m64 | F3 0F E6 /r | Convert Packed Doubleword Integers to Packed Double-Precision Floating-Point Values |
| CVTDQ2PS xmm1, xmm2/m128 | 0F 5B /r | Convert Packed Doubleword Integers to Packed Single-Precision Floating-Point Values |
| CVTPD2DQ xmm1, xmm2/m128 | F2 0F E6 /r | Convert Packed Double-Precision Floating-Point Values to Packed Doubleword Integers |
| CVTPD2PI mm, xmm/m128 | 66 0F 2D /r | Convert Packed Double-Precision FP Values to Packed Dword Integers |
| CVTPD2PS xmm1, xmm2/m128 | 66 0F 5A /r | Convert Packed Double-Precision Floating-Point Values to Packed Single-Precision Floating-Point Values |
| CVTPI2PD xmm, mm/m64 | 66 0F 2A /r | Convert Packed Dword Integers to Packed Double-Precision FP Values |
| CVTPS2DQ xmm1, xmm2/m128 | 66 0F 5B /r | Convert Packed Single-Precision Floating-Point Values to Packed Signed Doubleword Integer Values |
| CVTPS2PD xmm1, xmm2/m64 | 0F 5A /r | Convert Packed Single-Precision Floating-Point Values to Packed Double-Precision Floating-Point Values |
| CVTSD2SI r32, xmm1/m64 | F2 0F 2D /r | Convert Scalar Double-Precision Floating-Point Value to Doubleword Integer |
| CVTSD2SI r64, xmm1/m64 | F2 REX.W 0F 2D /r | Convert Scalar Double-Precision Floating-Point Value to Quadword Integer With Sign Extension |
| CVTSD2SS xmm1, xmm2/m64 | F2 0F 5A /r | Convert Scalar Double-Precision Floating-Point Value to Scalar Single-Precision Floating-Point Value |
| CVTSI2SD xmm1, r32/m32 | F2 0F 2A /r | Convert Doubleword Integer to Scalar Double-Precision Floating-Point Value |
| CVTSI2SD xmm1, r/m64 | F2 REX.W 0F 2A /r | Convert Quadword Integer to Scalar Double-Precision Floating-Point value |
| CVTSS2SD xmm1, xmm2/m32 | F3 0F 5A /r | Convert Scalar Single-Precision Floating-Point Value to Scalar Double-Precision Floating-Point Value |
| CVTTPD2DQ xmm1, xmm2/m128 | 66 0F E6 /r | Convert with Truncation Packed Double-Precision Floating-Point Values to Packed Doubleword Integers |
| CVTTPD2PI mm, xmm/m128 | 66 0F 2C /r | Convert with Truncation Packed Double-Precision FP Values to Packed Dword Integers |
| CVTTPS2DQ xmm1, xmm2/m128 | F3 0F 5B /r | Convert with Truncation Packed Single-Precision Floating-Point Values to Packed Signed Doubleword Integer Values |
| CVTTSD2SI r32, xmm1/m64 | F2 0F 2C /r | Convert with Truncation Scalar Double-Precision Floating-Point Value to Signed Dword Integer |
| CVTTSD2SI r64, xmm1/m64 | F2 REX.W 0F 2C /r | Convert with Truncation Scalar Double-Precision Floating-Point Value To Signed Qword Integer |

* CMPSD *and* MOVSD *have the same name as the*[*string*](https://en.wikipedia.org/wiki/String_(computer_science))*instruction mnemonics* CMPSD (CMPS) *and* MOVSD (MOVS)*; however, the former refer to scalar*[*double-precision*](https://en.wikipedia.org/wiki/Double_precision)[*floating-points*](https://en.wikipedia.org/wiki/Floating_point)*whereas the latters refer to [doubleword](https://en.wikipedia.org/wiki/Integer_(computer_science)" \o "Integer (computer science)) strings.*

**SSE2 SIMD integer instructions**[[edit](https://en.wikipedia.org/w/index.php?title=X86_instruction_listings&action=edit&section=53)]

**SSE2 MMX-like instructions extended to SSE registers**[[edit](https://en.wikipedia.org/w/index.php?title=X86_instruction_listings&action=edit&section=54)]

SSE2 allows execution of MMX instructions on SSE registers, processing twice the amount of data at once.

|  |  |  |
| --- | --- | --- |
| **Instruction** | **Opcode** | **Meaning** |
| MOVD xmm, r/m32 | 66 0F 6E /r | Move doubleword |
| MOVD r/m32, xmm | 66 0F 7E /r | Move doubleword |
| MOVQ xmm1, xmm2/m64 | F3 0F 7E /r | Move quadword |
| MOVQ xmm2/m64, xmm1 | 66 0F D6 /r | Move quadword |
| MOVQ r/m64, xmm | 66 REX.W 0F 7E /r | Move quadword |
| MOVQ xmm, r/m64 | 66 REX.W 0F 6E /r | Move quadword |
| PMOVMSKB reg, xmm | 66 0F D7 /r | Move a byte mask, zeroing the upper bits of the register |
| PEXTRW reg, xmm, imm8 | 66 0F C5 /r ib | Extract specified word and move it to reg, setting bits 15-0 and zeroing the rest |
| PINSRW xmm, r32/m16, imm8 | 66 0F C4 /r ib | Move low word at the specified word position |
| PACKSSDW xmm1, xmm2/m128 | 66 0F 6B /r | Converts 4 packed signed doubleword integers into 8 packed signed word integers with saturation |
| PACKSSWB xmm1, xmm2/m128 | 66 0F 63 /r | Converts 8 packed signed word integers into 16 packed signed byte integers with saturation |
| PACKUSWB xmm1, xmm2/m128 | 66 0F 67 /r | Converts 8 signed word integers into 16 unsigned byte integers with saturation |
| PADDB xmm1, xmm2/m128 | 66 0F FC /r | Add packed byte integers |
| PADDW xmm1, xmm2/m128 | 66 0F FD /r | Add packed word integers |
| PADDD xmm1, xmm2/m128 | 66 0F FE /r | Add packed doubleword integers |
| PADDQ xmm1, xmm2/m128 | 66 0F D4 /r | Add packed quadword integers. |
| PADDSB xmm1, xmm2/m128 | 66 0F EC /r | Add packed signed byte integers with saturation |
| PADDSW xmm1, xmm2/m128 | 66 0F ED /r | Add packed signed word integers with saturation |
| PADDUSB xmm1, xmm2/m128 | 66 0F DC /r | Add packed unsigned byte integers with saturation |
| PADDUSW xmm1, xmm2/m128 | 66 0F DD /r | Add packed unsigned word integers with saturation |
| PAND xmm1, xmm2/m128 | 66 0F DB /r | Bitwise AND |
| PANDN xmm1, xmm2/m128 | 66 0F DF /r | Bitwise AND NOT |
| POR xmm1, xmm2/m128 | 66 0F EB /r | Bitwise OR |
| PXOR xmm1, xmm2/m128 | 66 0F EF /r | Bitwise XOR |
| PCMPEQB xmm1, xmm2/m128 | 66 0F 74 /r | Compare packed bytes for equality. |
| PCMPEQW xmm1, xmm2/m128 | 66 0F 75 /r | Compare packed words for equality. |
| PCMPEQD xmm1, xmm2/m128 | 66 0F 76 /r | Compare packed doublewords for equality. |
| PCMPGTB xmm1, xmm2/m128 | 66 0F 64 /r | Compare packed signed byte integers for greater than |
| PCMPGTW xmm1, xmm2/m128 | 66 0F 65 /r | Compare packed signed word integers for greater than |
| PCMPGTD xmm1, xmm2/m128 | 66 0F 66 /r | Compare packed signed doubleword integers for greater than |
| PMULLW xmm1, xmm2/m128 | 66 0F D5 /r | Multiply packed signed word integers with saturation |
| PMULHW xmm1, xmm2/m128 | 66 0F E5 /r | Multiply the packed signed word integers, store the high 16 bits of the results |
| PMULHUW xmm1, xmm2/m128 | 66 0F E4 /r | Multiply packed unsigned word integers, store the high 16 bits of the results |
| PMULUDQ xmm1, xmm2/m128 | 66 0F F4 /r | Multiply packed unsigned doubleword integers |
| PSLLW xmm1, xmm2/m128 | 66 0F F1 /r | Shift words left while shifting in 0s |
| PSLLW xmm1, imm8 | 66 0F 71 /6 ib | Shift words left while shifting in 0s |
| PSLLD xmm1, xmm2/m128 | 66 0F F2 /r | Shift doublewords left while shifting in 0s |
| PSLLD xmm1, imm8 | 66 0F 72 /6 ib | Shift doublewords left while shifting in 0s |
| PSLLQ xmm1, xmm2/m128 | 66 0F F3 /r | Shift quadwords left while shifting in 0s |
| PSLLQ xmm1, imm8 | 66 0F 73 /6 ib | Shift quadwords left while shifting in 0s |
| PSRAD xmm1, xmm2/m128 | 66 0F E2 /r | Shift doubleword right while shifting in sign bits |
| PSRAD xmm1, imm8 | 66 0F 72 /4 ib | Shift doublewords right while shifting in sign bits |
| PSRAW xmm1, xmm2/m128 | 66 0F E1 /r | Shift words right while shifting in sign bits |
| PSRAW xmm1, imm8 | 66 0F 71 /4 ib | Shift words right while shifting in sign bits |
| PSRLW xmm1, xmm2/m128 | 66 0F D1 /r | Shift words right while shifting in 0s |
| PSRLW xmm1, imm8 | 66 0F 71 /2 ib | Shift words right while shifting in 0s |
| PSRLD xmm1, xmm2/m128 | 66 0F D2 /r | Shift doublewords right while shifting in 0s |
| PSRLD xmm1, imm8 | 66 0F 72 /2 ib | Shift doublewords right while shifting in 0s |
| PSRLQ xmm1, xmm2/m128 | 66 0F D3 /r | Shift quadwords right while shifting in 0s |
| PSRLQ xmm1, imm8 | 66 0F 73 /2 ib | Shift quadwords right while shifting in 0s |
| PSUBB xmm1, xmm2/m128 | 66 0F F8 /r | Subtract packed byte integers |
| PSUBW xmm1, xmm2/m128 | 66 0F F9 /r | Subtract packed word integers |
| PSUBD xmm1, xmm2/m128 | 66 0F FA /r | Subtract packed doubleword integers |
| PSUBQ xmm1, xmm2/m128 | 66 0F FB /r | Subtract packed quadword integers. |
| PSUBSB xmm1, xmm2/m128 | 66 0F E8 /r | Subtract packed signed byte integers with saturation |
| PSUBSW xmm1, xmm2/m128 | 66 0F E9 /r | Subtract packed signed word integers with saturation |
| PMADDWD xmm1, xmm2/m128 | 66 0F F5 /r | Multiply the packed word integers, add adjacent doubleword results |
| PSUBUSB xmm1, xmm2/m128 | 66 0F D8 /r | Subtract packed unsigned byte integers with saturation |
| PSUBUSW xmm1, xmm2/m128 | 66 0F D9 /r | Subtract packed unsigned word integers with saturation |
| PUNPCKHBW xmm1, xmm2/m128 | 66 0F 68 /r | Unpack and interleave high-order bytes |
| PUNPCKHWD xmm1, xmm2/m128 | 66 0F 69 /r | Unpack and interleave high-order words |
| PUNPCKHDQ xmm1, xmm2/m128 | 66 0F 6A /r | Unpack and interleave high-order doublewords |
| PUNPCKLBW xmm1, xmm2/m128 | 66 0F 60 /r | Interleave low-order bytes |
| PUNPCKLWD xmm1, xmm2/m128 | 66 0F 61 /r | Interleave low-order words |
| PUNPCKLDQ xmm1, xmm2/m128 | 66 0F 62 /r | Interleave low-order doublewords |
| PAVGB xmm1, xmm2/m128 | 66 0F E0, /r | Average packed unsigned byte integers with rounding |
| PAVGW xmm1, xmm2/m128 | 66 0F E3 /r | Average packed unsigned word integers with rounding |
| PMINUB xmm1, xmm2/m128 | 66 0F DA /r | Compare packed unsigned byte integers and store packed minimum values |
| PMINSW xmm1, xmm2/m128 | 66 0F EA /r | Compare packed signed word integers and store packed minimum values |
| PMAXSW xmm1, xmm2/m128 | 66 0F EE /r | Compare packed signed word integers and store maximum packed values |
| PMAXUB xmm1, xmm2/m128 | 66 0F DE /r | Compare packed unsigned byte integers and store packed maximum values |
| PSADBW xmm1, xmm2/m128 | 66 0F F6 /r | Computes the absolute differences of the packed unsigned byte integers; the 8 low differences and 8 high differences are then summed separately to produce two unsigned word integer results |

**SSE2 integer instructions for SSE registers only**[[edit](https://en.wikipedia.org/w/index.php?title=X86_instruction_listings&action=edit&section=55)]

The following instructions can be used only on SSE registers, since by their nature they do not work on MMX registers

|  |  |  |
| --- | --- | --- |
| **Instruction** | **Opcode** | **Meaning** |
| MASKMOVDQU xmm1, xmm2 | 66 0F F7 /r | Non-Temporal Store of Selected Bytes from an XMM Register into Memory |
| MOVDQ2Q mm, xmm | F2 0F D6 /r | Move low quadword from XMM to MMX register. |
| MOVDQA xmm1, xmm2/m128 | 66 0F 6F /r | Move aligned double quadword |
| MOVDQA xmm2/m128, xmm1 | 66 0F 7F /r | Move aligned double quadword |
| MOVDQU xmm1, xmm2/m128 | F3 0F 6F /r | Move unaligned double quadword |
| MOVDQU xmm2/m128, xmm1 | F3 0F 7F /r | Move unaligned double quadword |
| MOVQ2DQ xmm, mm | F3 0F D6 /r | Move quadword from MMX register to low quadword of XMM register |
| MOVNTDQ m128, xmm1 | 66 0F E7 /r | Store Packed Integers Using Non-Temporal Hint |
| PSHUFHW xmm1, xmm2/m128, imm8 | F3 0F 70 /r ib | Shuffle packed high words. |
| PSHUFLW xmm1, xmm2/m128, imm8 | F2 0F 70 /r ib | Shuffle packed low words. |
| PSHUFD xmm1, xmm2/m128, imm8 | 66 0F 70 /r ib | Shuffle packed doublewords. |
| PSLLDQ xmm1, imm8 | 66 0F 73 /7 ib | Packed shift left logical double quadwords. |
| PSRLDQ xmm1, imm8 | 66 0F 73 /3 ib | Packed shift right logical double quadwords. |
| PUNPCKHQDQ xmm1, xmm2/m128 | 66 0F 6D /r | Unpack and interleave high-order quadwords, |
| PUNPCKLQDQ xmm1, xmm2/m128 | 66 0F 6C /r | Interleave low quadwords, |

[**SSE3**](https://en.wikipedia.org/wiki/SSE3)**instructions**[[edit](https://en.wikipedia.org/w/index.php?title=X86_instruction_listings&action=edit&section=56)]

*Added with Pentium 4 supporting SSE3*

**SSE3 SIMD floating-point instructions**[[edit](https://en.wikipedia.org/w/index.php?title=X86_instruction_listings&action=edit&section=57)]

|  |  |  |  |
| --- | --- | --- | --- |
| **Instruction** | **Opcode** | **Meaning** | **Notes** |
| ADDSUBPS xmm1, xmm2/m128 | F2 0F D0 /r | Add/subtract single-precision floating-point values | for Complex Arithmetic |
| ADDSUBPD xmm1, xmm2/m128 | 66 0F D0 /r | Add/subtract double-precision floating-point values |
| [MOVDDUP](https://en.wikipedia.org/wiki/MOVDDUP) xmm1, xmm2/m64 | F2 0F 12 /r | Move double-precision floating-point value and duplicate |
| MOVSLDUP xmm1, xmm2/m128 | F3 0F 12 /r | Move and duplicate even index single-precision floating-point values |
| MOVSHDUP xmm1, xmm2/m128 | F3 0F 16 /r | Move and duplicate odd index single-precision floating-point values |
| HADDPS xmm1, xmm2/m128 | F2 0F 7C /r | Horizontal add packed single-precision floating-point values | for Graphics |
| HADDPD xmm1, xmm2/m128 | 66 0F 7C /r | Horizontal add packed double-precision floating-point values |
| HSUBPS xmm1, xmm2/m128 | F2 0F 7D /r | Horizontal subtract packed single-precision floating-point values |
| HSUBPD xmm1, xmm2/m128 | 66 0F 7D /r | Horizontal subtract packed double-precision floating-point values |

**SSE3 SIMD integer instructions**[[edit](https://en.wikipedia.org/w/index.php?title=X86_instruction_listings&action=edit&section=58)]

|  |  |  |  |
| --- | --- | --- | --- |
| **Instruction** | **Opcode** | **Meaning** | **Notes** |
| LDDQU xmm1, mem | F2 0F F0 /r | Load unaligned data and return double quadword | Instructionally equivalent to MOVDQU. For video encoding |

[**SSSE3**](https://en.wikipedia.org/wiki/SSSE3)**instructions**[[edit](https://en.wikipedia.org/w/index.php?title=X86_instruction_listings&action=edit&section=59)]

*Added with*[*Xeon*](https://en.wikipedia.org/wiki/Xeon)*5100 series and initial*[*Core 2*](https://en.wikipedia.org/wiki/Core_2)

The following MMX-like instructions extended to SSE registers were added with SSSE3

|  |  |  |
| --- | --- | --- |
| **Instruction** | **Opcode** | **Meaning** |
| PSIGNB xmm1, xmm2/m128 | 66 0F 38 08 /r | Negate/zero/preserve packed byte integers depending on corresponding sign |
| PSIGNW xmm1, xmm2/m128 | 66 0F 38 09 /r | Negate/zero/preserve packed word integers depending on corresponding sign |
| PSIGND xmm1, xmm2/m128 | 66 0F 38 0A /r | Negate/zero/preserve packed doubleword integers depending on corresponding |
| PSHUFB xmm1, xmm2/m128 | 66 0F 38 00 /r | Shuffle bytes |
| PMULHRSW xmm1, xmm2/m128 | 66 0F 38 0B /r | Multiply 16-bit signed words, scale and round signed doublewords, pack high 16 bits |
| PMADDUBSW xmm1, xmm2/m128 | 66 0F 38 04 /r | Multiply signed and unsigned bytes, add horizontal pair of signed words, pack saturated signed-words |
| PHSUBW xmm1, xmm2/m128 | 66 0F 38 05 /r | Subtract and pack 16-bit signed integers horizontally |
| PHSUBSW xmm1, xmm2/m128 | 66 0F 38 07 /r | Subtract and pack 16-bit signed integer horizontally with saturation |
| PHSUBD xmm1, xmm2/m128 | 66 0F 38 06 /r | Subtract and pack 32-bit signed integers horizontally |
| PHADDSW xmm1, xmm2/m128 | 66 0F 38 03 /r | Add and pack 16-bit signed integers horizontally with saturation |
| PHADDW xmm1, xmm2/m128 | 66 0F 38 01 /r | Add and pack 16-bit integers horizontally |
| PHADDD xmm1, xmm2/m128 | 66 0F 38 02 /r | Add and pack 32-bit integers horizontally |
| PALIGNR xmm1, xmm2/m128, imm8 | 66 0F 3A 0F /r ib | Concatenate destination and source operands, extract byte-aligned result shifted to the right |
| PABSB xmm1, xmm2/m128 | 66 0F 38 1C /r | Compute the absolute value of bytes and store unsigned result |
| PABSW xmm1, xmm2/m128 | 66 0F 38 1D /r | Compute the absolute value of 16-bit integers and store unsigned result |
| PABSD xmm1, xmm2/m128 | 66 0F 38 1E /r | Compute the absolute value of 32-bit integers and store unsigned result |

[**SSE4**](https://en.wikipedia.org/wiki/SSE4)**instructions**[[edit](https://en.wikipedia.org/w/index.php?title=X86_instruction_listings&action=edit&section=60)]

[**SSE4.1**](https://en.wikipedia.org/wiki/SSE4.1)[[edit](https://en.wikipedia.org/w/index.php?title=X86_instruction_listings&action=edit&section=61)]

*Added with*[*Core 2*](https://en.wikipedia.org/wiki/Core_2)*manufactured in*[*45nm*](https://en.wikipedia.org/wiki/45nm)

**SSE4.1 SIMD floating-point instructions**[[edit](https://en.wikipedia.org/w/index.php?title=X86_instruction_listings&action=edit&section=62)]

|  |  |  |
| --- | --- | --- |
| **Instruction** | **Opcode** | **Meaning** |
| DPPS xmm1, xmm2/m128, imm8 | 66 0F 3A 40 /r ib | Selectively multiply packed SP floating-point values, add and selectively store |
| DPPD xmm1, xmm2/m128, imm8 | 66 0F 3A 41 /r ib | Selectively multiply packed DP floating-point values, add and selectively store |
| BLENDPS xmm1, xmm2/m128, imm8 | 66 0F 3A 0C /r ib | Select packed single precision floating-point values from specified mask |
| BLENDVPS xmm1, xmm2/m128, <XMM0> | 66 0F 38 14 /r | Select packed single precision floating-point values from specified mask |
| BLENDPD xmm1, xmm2/m128, imm8 | 66 0F 3A 0D /r ib | Select packed DP-FP values from specified mask |
| BLENDVPD xmm1, xmm2/m128 , <XMM0> | 66 0F 38 15 /r | Select packed DP FP values from specified mask |
| ROUNDPS xmm1, xmm2/m128, imm8 | 66 0F 3A 08 /r ib | Round packed single precision floating-point values |
| ROUNDSS xmm1, xmm2/m32, imm8 | 66 0F 3A 0A /r ib | Round the low packed single precision floating-point value |
| ROUNDPD xmm1, xmm2/m128, imm8 | 66 0F 3A 09 /r ib | Round packed double precision floating-point values |
| ROUNDSD xmm1, xmm2/m64, imm8 | 66 0F 3A 0B /r ib | Round the low packed double precision floating-point value |
| INSERTPS xmm1, xmm2/m32, imm8 | 66 0F 3A 21 /r ib | Insert a selected single-precision floating-point value at the specified destination element and zero out destination elements |
| EXTRACTPS reg/m32, xmm1, imm8 | 66 0F 3A 17 /r ib | Extract one single-precision floating-point value at specified offset and store the result (zero-extended, if applicable) |

**SSE4.1 SIMD integer instructions**[[edit](https://en.wikipedia.org/w/index.php?title=X86_instruction_listings&action=edit&section=63)]

|  |  |  |
| --- | --- | --- |
| **Instruction** | **Opcode** | **Meaning** |
| MPSADBW xmm1, xmm2/m128, imm8 | 66 0F 3A 42 /r ib | Sums absolute 8-bit integer difference of adjacent groups of 4 byte integers with starting offset |
| PHMINPOSUW xmm1, xmm2/m128 | 66 0F 38 41 /r | Find the minimum unsigned word |
| PMULLD xmm1, xmm2/m128 | 66 0F 38 40 /r | Multiply the packed dword signed integers and store the low 32 bits |
| PMULDQ xmm1, xmm2/m128 | 66 0F 38 28 /r | Multiply packed signed doubleword integers and store quadword result |
| PBLENDVB xmm1, xmm2/m128, <XMM0> | 66 0F 38 10 /r | Select byte values from specified mask |
| PBLENDW xmm1, xmm2/m128, imm8 | 66 0F 3A 0E /r ib | Select words from specified mask |
| PMINSB xmm1, xmm2/m128 | 66 0F 38 38 /r | Compare packed signed byte integers |
| PMINUW xmm1, xmm2/m128 | 66 0F 38 3A/r | Compare packed unsigned word integers |
| PMINSD xmm1, xmm2/m128 | 66 0F 38 39 /r | Compare packed signed dword integers |
| PMINUD xmm1, xmm2/m128 | 66 0F 38 3B /r | Compare packed unsigned dword integers |
| PMAXSB xmm1, xmm2/m128 | 66 0F 38 3C /r | Compare packed signed byte integers |
| PMAXUW xmm1, xmm2/m128 | 66 0F 38 3E/r | Compare packed unsigned word integers |
| PMAXSD xmm1, xmm2/m128 | 66 0F 38 3D /r | Compare packed signed dword integers |
| PMAXUD xmm1, xmm2/m128 | 66 0F 38 3F /r | Compare packed unsigned dword integers |
| PINSRB xmm1, r32/m8, imm8 | 66 0F 3A 20 /r ib | Insert a byte integer value at specified destination element |
| PINSRD xmm1, r/m32, imm8 | 66 0F 3A 22 /r ib | Insert a dword integer value at specified destination element |
| PINSRQ xmm1, r/m64, imm8 | 66 REX.W 0F 3A 22 /r ib | Insert a qword integer value at specified destination element |
| PEXTRB reg/m8, xmm2, imm8 | 66 0F 3A 14 /r ib | Extract a byte integer value at source byte offset, upper bits are zeroed. |
| PEXTRW reg/m16, xmm, imm8 | 66 0F 3A 15 /r ib | Extract word and copy to lowest 16 bits, zero-extended |
| PEXTRD r/m32, xmm2, imm8 | 66 0F 3A 16 /r ib | Extract a dword integer value at source dword offset |
| PEXTRQ r/m64, xmm2, imm8 | 66 REX.W 0F 3A 16 /r ib | Extract a qword integer value at source qword offset |
| PMOVSXBW xmm1, xmm2/m64 | 66 0f 38 20 /r | Sign extend 8 packed 8-bit integers to 8 packed 16-bit integers |
| PMOVZXBW xmm1, xmm2/m64 | 66 0f 38 30 /r | Zero extend 8 packed 8-bit integers to 8 packed 16-bit integers |
| PMOVSXBD xmm1, xmm2/m32 | 66 0f 38 21 /r | Sign extend 4 packed 8-bit integers to 4 packed 32-bit integers |
| PMOVZXBD xmm1, xmm2/m32 | 66 0f 38 31 /r | Zero extend 4 packed 8-bit integers to 4 packed 32-bit integers |
| PMOVSXBQ xmm1, xmm2/m16 | 66 0f 38 22 /r | Sign extend 2 packed 8-bit integers to 2 packed 64-bit integers |
| PMOVZXBQ xmm1, xmm2/m16 | 66 0f 38 32 /r | Zero extend 2 packed 8-bit integers to 2 packed 64-bit integers |
| PMOVSXWD xmm1, xmm2/m64 | 66 0f 38 23/r | Sign extend 4 packed 16-bit integers to 4 packed 32-bit integers |
| PMOVZXWD xmm1, xmm2/m64 | 66 0f 38 33 /r | Zero extend 4 packed 16-bit integers to 4 packed 32-bit integers |
| PMOVSXWQ xmm1, xmm2/m32 | 66 0f 38 24 /r | Sign extend 2 packed 16-bit integers to 2 packed 64-bit integers |
| PMOVZXWQ xmm1, xmm2/m32 | 66 0f 38 34 /r | Zero extend 2 packed 16-bit integers to 2 packed 64-bit integers |
| PMOVSXDQ xmm1, xmm2/m64 | 66 0f 38 25 /r | Sign extend 2 packed 32-bit integers to 2 packed 64-bit integers |
| PMOVZXDQ xmm1, xmm2/m64 | 66 0f 38 35 /r | Zero extend 2 packed 32-bit integers to 2 packed 64-bit integers |
| PTEST xmm1, xmm2/m128 | 66 0F 38 17 /r | Set ZF if AND result is all 0s, set CF if AND NOT result is all 0s |
| PCMPEQQ xmm1, xmm2/m128 | 66 0F 38 29 /r | Compare packed qwords for equality |
| PACKUSDW xmm1, xmm2/m128 | 66 0F 38 2B /r | Convert 2 × 4 packed signed doubleword integers into 8 packed unsigned word integers with saturation |
| MOVNTDQA xmm1, m128 | 66 0F 38 2A /r | Move double quadword using non-temporal hint if WC memory type |

[**SSE4a**](https://en.wikipedia.org/wiki/SSE4a)[[edit](https://en.wikipedia.org/w/index.php?title=X86_instruction_listings&action=edit&section=64)]

*Added with [Phenom](https://en.wikipedia.org/wiki/Phenom_(processor)" \o "Phenom (processor)) processors*

* EXTRQ/INSERTQ
* MOVNTSD/MOVNTSS

[**SSE4.2**](https://en.wikipedia.org/wiki/SSE4.2)[[edit](https://en.wikipedia.org/w/index.php?title=X86_instruction_listings&action=edit&section=65)]

*Added with*[*Nehalem*](https://en.wikipedia.org/wiki/Nehalem_(microarchitecture))*processors*

|  |  |  |
| --- | --- | --- |
| **Instruction** | **Opcode** | **Meaning** |
| PCMPESTRI xmm1, xmm2/m128, imm8 | 66 0F 3A 61 /r imm8 | Packed comparison of string data with explicit lengths, generating an index |
| PCMPESTRM xmm1, xmm2/m128, imm8 | 66 0F 3A 60 /r imm8 | Packed comparison of string data with explicit lengths, generating a mask |
| PCMPISTRI xmm1, xmm2/m128, imm8 | 66 0F 3A 63 /r imm8 | Packed comparison of string data with implicit lengths, generating an index |
| PCMPISTRM xmm1, xmm2/m128, imm8 | 66 0F 3A 62 /r imm8 | Packed comparison of string data with implicit lengths, generating a mask |
| PCMPGTQ xmm1,xmm2/m128 | 66 0F 38 37 /r | Compare packed signed qwords for greater than. |

[**SSE5**](https://en.wikipedia.org/wiki/SSE5)**derived instructions**[[edit](https://en.wikipedia.org/w/index.php?title=X86_instruction_listings&action=edit&section=66)]

SSE5 was a proposed SSE extension by AMD. The bundle did not include the full set of Intel's SSE4 instructions, making it a competitor to SSE4 rather than a successor. AMD chose not to implement SSE5 as originally proposed, however, derived SSE extensions were introduced.

[**XOP**](https://en.wikipedia.org/wiki/XOP_instruction_set)[[edit](https://en.wikipedia.org/w/index.php?title=X86_instruction_listings&action=edit&section=67)]

Introduced with the bulldozer processor core, removed again from [Zen (microarchitecture)](https://en.wikipedia.org/wiki/Zen_(microarchitecture)) onward.

A revision of most of the SSE5 instruction set

[**F16C**](https://en.wikipedia.org/wiki/F16C)[[edit](https://en.wikipedia.org/w/index.php?title=X86_instruction_listings&action=edit&section=68)]

Half-precision floating-point conversion.

[**FMA3**](https://en.wikipedia.org/wiki/FMA3)[[edit](https://en.wikipedia.org/w/index.php?title=X86_instruction_listings&action=edit&section=69)]

Supported in AMD processors starting with the Piledriver architecture and Intel starting with Haswell processors and Broadwell processors since 2014.

[Fused multiply-add](https://en.wikipedia.org/wiki/Fused_multiply-add) (floating-point vector multiply–accumulate) with three operands.

[**FMA4**](https://en.wikipedia.org/wiki/FMA_instruction_set)[[edit](https://en.wikipedia.org/w/index.php?title=X86_instruction_listings&action=edit&section=70)]

Supported in AMD processors starting with the Bulldozer architecture. Not supported by any intel chip as of 2017.

[Fused multiply-add](https://en.wikipedia.org/wiki/Fused_multiply-add) with four operands. FMA4 was realized in hardware before FMA3.

|  |  |  |  |
| --- | --- | --- | --- |
| **Instruction** | **Opcode** | **Meaning** | **Notes** |
| VFMADDPD xmm0, xmm1, xmm2, xmm3 | C4E3 WvvvvL01 69 /r /is4 | Fused Multiply-Add of Packed Double-Precision Floating-Point Values |  |
| VFMADDPS xmm0, xmm1, xmm2, xmm3 | C4E3 WvvvvL01 68 /r /is4 | Fused Multiply-Add of Packed Single-Precision Floating-Point Values |  |
| VFMADDSD xmm0, xmm1, xmm2, xmm3 | C4E3 WvvvvL01 6B /r /is4 | Fused Multiply-Add of Scalar Double-Precision Floating-Point Values |  |
| VFMADDSS xmm0, xmm1, xmm2, xmm3 | C4E3 WvvvvL01 6A /r /is4 | Fused Multiply-Add of Scalar Single-Precision Floating-Point Values |  |
| VFMADDSUBPD xmm0, xmm1, xmm2, xmm3 | C4E3 WvvvvL01 5D /r /is4 | Fused Multiply-Alternating Add/Subtract of Packed Double-Precision Floating-Point Values |  |
| VFMADDSUBPS xmm0, xmm1, xmm2, xmm3 | C4E3 WvvvvL01 5C /r /is4 | Fused Multiply-Alternating Add/Subtract of Packed Single-Precision Floating-Point Values |  |
| VFMSUBADDPD xmm0, xmm1, xmm2, xmm3 | C4E3 WvvvvL01 5F /r /is4 | Fused Multiply-Alternating Subtract/Add of Packed Double-Precision Floating-Point Values |  |
| VFMSUBADDPS xmm0, xmm1, xmm2, xmm3 | C4E3 WvvvvL01 5E /r /is4 | Fused Multiply-Alternating Subtract/Add of Packed Single-Precision Floating-Point Values |  |
| VFMSUBPD xmm0, xmm1, xmm2, xmm3 | C4E3 WvvvvL01 6D /r /is4 | Fused Multiply-Subtract of Packed Double-Precision Floating-Point Values |  |
| VFMSUBPS xmm0, xmm1, xmm2, xmm3 | C4E3 WvvvvL01 6C /r /is4 | Fused Multiply-Subtract of Packed Single-Precision Floating-Point Values |  |
| VFMSUBSD xmm0, xmm1, xmm2, xmm3 | C4E3 WvvvvL01 6F /r /is4 | Fused Multiply-Subtract of Scalar Double-Precision Floating-Point Values |  |
| VFMSUBSS xmm0, xmm1, xmm2, xmm3 | C4E3 WvvvvL01 6E /r /is4 | Fused Multiply-Subtract of Scalar Single-Precision Floating-Point Values |  |
| VFNMADDPD xmm0, xmm1, xmm2, xmm3 | C4E3 WvvvvL01 79 /r /is4 | Fused Negative Multiply-Add of Packed Double-Precision Floating-Point Values |  |
| VFNMADDPS xmm0, xmm1, xmm2, xmm3 | C4E3 WvvvvL01 78 /r /is4 | Fused Negative Multiply-Add of Packed Single-Precision Floating-Point Values |  |
| VFNMADDSD xmm0, xmm1, xmm2, xmm3 | C4E3 WvvvvL01 7B /r /is4 | Fused Negative Multiply-Add of Scalar Double-Precision Floating-Point Values |  |
| VFNMADDSS xmm0, xmm1, xmm2, xmm3 | C4E3 WvvvvL01 7A /r /is4 | Fused Negative Multiply-Add of Scalar Single-Precision Floating-Point Values |  |
| VFNMSUBPD xmm0, xmm1, xmm2, xmm3 | C4E3 WvvvvL01 7D /r /is4 | Fused Negative Multiply-Subtract of Packed Double-Precision Floating-Point Values |  |
| VFNMSUBPS xmm0, xmm1, xmm2, xmm3 | C4E3 WvvvvL01 7C /r /is4 | Fused Negative Multiply-Subtract of Packed Single-Precision Floating-Point Values |  |
| VFNMSUBSD xmm0, xmm1, xmm2, xmm3 | C4E3 WvvvvL01 7F /r /is4 | Fused Negative Multiply-Subtract of Scalar Double-Precision Floating-Point Values |  |
| VFNMSUBSS xmm0, xmm1, xmm2, xmm3 | C4E3 WvvvvL01 7E /r /is4 | Fused Negative Multiply-Subtract of Scalar Single-Precision Floating-Point Values |  |

[**AVX**](https://en.wikipedia.org/wiki/AVX)[[edit](https://en.wikipedia.org/w/index.php?title=X86_instruction_listings&action=edit&section=71)]

First supported by Intel with Sandy Bridge and by AMD with Bulldozer.

Vector operations on 256 bit registers.

[**AVX2**](https://en.wikipedia.org/wiki/AVX2)[[edit](https://en.wikipedia.org/w/index.php?title=X86_instruction_listings&action=edit&section=72)]

Introduced in Intel's Haswell microarchitecture and AMD's Excavator.

Expansion of most vector integer SSE and AVX instructions to 256 bits

[**AVX-512**](https://en.wikipedia.org/wiki/AVX-512)[[edit](https://en.wikipedia.org/w/index.php?title=X86_instruction_listings&action=edit&section=73)]

Introduced in Intel's Xeon Phi x200

Vector operations on 512 bit registers.

Cryptographic instructions[[edit](https://en.wikipedia.org/w/index.php?title=X86_instruction_listings&action=edit&section=74)]

**Intel AES instructions**[[edit](https://en.wikipedia.org/w/index.php?title=X86_instruction_listings&action=edit&section=75)]

*Main article:*[*AES instruction set*](https://en.wikipedia.org/wiki/AES_instruction_set)

6 new instructions.

|  |  |
| --- | --- |
| **Instruction** | **Description** |
| AESENC | Perform one round of an [AES](https://en.wikipedia.org/wiki/Advanced_Encryption_Standard) encryption flow |
| AESENCLAST | Perform the last round of an AES encryption flow |
| AESDEC | Perform one round of an AES decryption flow |
| AESDECLAST | Perform the last round of an AES decryption flow |
| AESKEYGENASSIST | Assist in AES round key generation |
| AESIMC | Assist in AES Inverse Mix Columns |

**Intel SHA instructions**[[edit](https://en.wikipedia.org/w/index.php?title=X86_instruction_listings&action=edit&section=76)]

*Main article:*[*Intel SHA extensions*](https://en.wikipedia.org/wiki/Intel_SHA_extensions)

7 new instructions.

|  |  |
| --- | --- |
| **Instruction** | **Description** |
| SHA1RNDS4 |  |
| SHA1NEXTE |  |
| SHA1MSG1 |  |
| SHA1MSG2 |  |
| SHA256RNDS2 |  |
| SHA256MSG1 |  |
| SHA256MSG2 |  |

Undocumented instructions[[edit](https://en.wikipedia.org/w/index.php?title=X86_instruction_listings&action=edit&section=77)]

**Undocumented x86 instructions**[[edit](https://en.wikipedia.org/w/index.php?title=X86_instruction_listings&action=edit&section=78)]

The x86 CPUs contain [undocumented instructions](https://en.wikipedia.org/wiki/Illegal_opcode) which are implemented on the chips but not listed in some official documents. They can be found in various sources across the Internet, such as [Ralf Brown's Interrupt List](https://en.wikipedia.org/wiki/Ralf_Brown%27s_Interrupt_List) and at [sandpile.org](http://www.sandpile.org/)

|  |  |  |  |
| --- | --- | --- | --- |
| **Mnemonic** | **Opcode** | **Description** | **Status** |
| AAM imm8 | D4 imm8 | Divide AL by imm8, put the quotient in AH, and the remainder in AL | Available beginning with 8086, documented since Pentium (earlier documentation lists no arguments) |
| AAD imm8 | D5 imm8 | Multiplication counterpart of AAM | Available beginning with 8086, documented since Pentium (earlier documentation lists no arguments) |
| SALC | D6 | Set AL depending on the value of the Carry Flag (a 1-byte alternative of SBB AL, AL) | Available beginning with 8086, but only documented since Pentium Pro. |
| ICEBP | F1 | Single byte single-step exception / Invoke [ICE](https://en.wikipedia.org/wiki/In-circuit_emulator) | Available beginning with 80386, documented (as INT1) since Pentium Pro |
| Unknown mnemonic | 0F 04 | Exact purpose unknown, causes CPU hang ([HCF](https://en.wikipedia.org/wiki/Halt_and_Catch_Fire)). The only way out is CPU reset.[[5]](https://en.wikipedia.org/wiki/X86_instruction_listings#cite_note-5)  In some implementations, emulated through [BIOS](https://en.wikipedia.org/wiki/BIOS) as a [halting](https://en.wikipedia.org/wiki/HLT_(x86_instruction)) sequence.[[6]](https://en.wikipedia.org/wiki/X86_instruction_listings#cite_note-6) | Only available on 80286 |
| [LOADALL](https://en.wikipedia.org/wiki/LOADALL) | 0F 05 | Loads All Registers from Memory Address 0x000800H | Only available on 80286 |
| [LOADALLD](https://en.wikipedia.org/wiki/LOADALLD) | 0F 07 | Loads All Registers from Memory Address ES:EDI | Only available on 80386 |
| UD1 | 0F B9 | Intentionally undefined instruction, but unlike UD2 this was not published |  |