x86 instruction listings

From Wikipedia, the free encyclopedia

The x86 instruction set has been extended several times, introducing wider registers and datatypes and/or new functionality[1]

Contents

x86 integer instructions

This is the full 8086/8088 instruction set, but 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 for a quick tutorial for this processor family. The updated instruction set is also grouped according to architecture (i386, i486, i686) and more generally is referred to as x86_32 and x86_64 (also known as AMD64).

Original 8086/8088 instructions

Original 8086/8088 instruction set

Instruction	Meaning	Notes	Opcode
AAA	ASCII adjust AL after addition	used with unpacked binary coded decimal Example: ;Assume AL=0011 0101,ASCII 5; BL=0011 1001,ASCII 9 ||	
AAD	ASCII adjust AX before division	8086/8088 datasheet documents only base 10 version of the AAD instruction (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	
AAM	ASCII adjust AX after multiplication	Only base 10 version is documented, see notes for AAD	
AAS	ASCII adjust AL after subtraction		
ADC	Add with carry	destination := destination + source + carry_flag	
ADD	Add	(1) r/m += r/imm; (2) r += m/imm;	
AND	Logical AND	(1) r/m &= r/imm; (2) r &= m/imm;	
CALL	Call procedure	push eip + 2 ; jmp operand	
CBW	Convert byte to word		
CLC	Clear carry flag	CF = 0;	
CLD	Clear direction flag	DF = 0;	
CLI	Clear interrupt flag	IF = 0;	
CMC	Complement carry flag		
CMP	Compare operands		
CMPSB	Compare bytes in memory		
CMPSW	Compare words		
CWD	Convert word to doubleword		
DAA	Decimal adjust AL after addition	(used with packed binary coded decimal)	
DAS	Decimal adjust AL after subtraction		
DEC	Decrement by 1		
DIV	Unsigned divide	DX:AX = DX:AX / r/m; resulting DX = remainder	
ESC	Used with floating-point unit		
HLT	Enter halt state		0xF4
IDIV	Signed divide	DX:AX = DX:AX / r/m; resulting DX = remainder	
IMUL	Signed multiply	(1) DX:AX = AX * r/m; (2) AX = AL * r/m	
IN	Input from port	(1) AL = port[imm]; (2) AL = port[DX]; (3) AX = port[DX];	
INC	Increment by 1		
INT	Call to interrupt		
INTO	Call to interrupt if overflow		

IRET	Return from interrupt		
Jcc	Jump if condition	(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)	
JCXZ	Jump if CX is zero		
JMP	Jump		
LAHF	Load flags into AH register		
LDS	Load pointer using DS		
LEA	Load Effective Address		
LES	Load ES with pointer		
LOCK	Assert BUS LOCK# signal	(for multiprocessing)	
LODSB	Load string byte	if (DF==0) AL = *SI++; else AL = *SI--;	
LODSW	Load string word	if (DF==0) AX = *SI++; else AX = *SI--;	
LOOP/LOOPx	Loop control	(LOOPE, LOOPNE, LOOPNZ, LOOPZ) if (x && --CX) goto lbl;	
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--;	
MOVSW	Move word from string to string	if (DF==0) *(word*)DI++ = *(word*)SI++; else *(word*)DI-- = *(word*)SI--;	
MUL	Unsigned multiply	(1) DX:AX = AX * r/m; (2) AX = AL * r/m;	
NEG	Two's complement negation	r/m *= -1;	
NOP	No operation	opcode equivalent to XCHG EAX, EAX	0x90
NOT	Negate the operand, logical NOT	r/m ^= -1;	
OR	Logical OR	= 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	*SP++ = r/m; POP CS (opcode 0x0F) works only on 8086/8088. Later CPUs use 0x0F as a prefix for newer instructions.	0x0F
POPF	Pop data from flags register	*SP++ = flags;	
PUSH	Push data onto stack	*--SP = r/m;	
PUSHF	Push flags onto stack	*--SP = flags;	
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		
SAL	Shift Arithmetically 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 SALC instruction	
SCASB	Compare byte string		

SCASW	Compare word string		
SHL	Shift left (unsigned shift left)		
SHR	Shift right (unsigned shift right)		
STC	Set carry flag	CF = 1;	
STD	Set direction flag	DF = 1;	
STI	Set interrupt flag	IF = 1;	
STOSB	Store byte in string	if (DF==0) *ES:DI++ = AL; else *ES:DI-- = AL;	
STOSW	Store word in string	if (DF==0) *ES:DI++ = AX; else *ES:DI-- = AX;	
SUB	Subtraction	(1) r/m -= r/imm; (2) r -= m/imm;	
TEST	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)	
XCHG	Exchange data	r :=: r/m;	
XLAT	Table look-up translation	behaves like `MOV AL, [BX+AL]`	
XOR	Exclusive OR	(1) r/m ^= r/imm; (2) r ^= m/imm;	

Added in specific processors

Added with 80186/80188

Instruction	Meaning	Notes
BOUND	Check array index against bounds	raises software interrupt 5 if test fails

ENTER	Enter stack frame	equivalent to
		`Push (E)BP`
		`Set a temporary value (E)FRAME_PTR:=(E)SP`
		`If LEVEL > 0 then`
		`  Repeat (LEVEL - 1) times:`
		`  (E)BP:=(E)BP - 2`
		`  Push the word pointed to by (E)BP`
		`  End repeat`
		`  Push (E)FRAME_PTR`
		`End if`
		`(E)BP:=(E)FRAME_PTR`
		`(E)SP:=(E)SP - first operand.`

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	equivalent to
		`MOV SP, BP`
		`POP BP`

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`
		`PUSH BP`
		`PUSH SI`
		`PUSH DI`

Added with 80286

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	Load all CPU registers, including internal ones such as GDT	Undocumented, 80286 and 80386 only
LSL	Load segment limit	
LTR	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

Instruction	Meaning	Notes
BSF	Bit scan forward	
BSR	Bit scan reverse	
BT	Bit test	
BTC	Bit test and complement	
BTR	Bit test and reset	
BTS	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, LOOPccW	Loop, conditional loop	Same as LOOP, LOOPcc 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)
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

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
INVD	Invalidate Internal Caches	Flush internal caches
INVLPG	Invalidate TLB 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

Instruction
CPUID
CMPXCHG8B
RDMSR
RDTSC
WRMSR
RSM [1] (http://www.softeng.rl.ac.uk/st/archive/SoftEng/SESP/html/SoftwareTools/vtune/users_guide/mergedProjects/analyzer_ec/mergedProjects/reference_olh/mergedProjects/instructio

Added with Pentium MMX

Instruction	Meaning	Notes
RDPMC	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

Instruction	Meaning	Notes
SYSCALL		
SYSRET		functionally equivalent to SYSENTER and SYSEXIT

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

Added with Pentium Pro

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)
SYSENTER	SYStem call ENTER	
SYSEXIT	SYStem call EXIT	
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 SSE

Instruction	Meaning	Notes
MASKMOVQ	Masked Move of Quadword	Selectively write bytes from mm1 to memory location using the byte mask in mm2
MOVNTPS	Move Aligned Four Packed Single-FP Non Temporal	Move packed single-precision floating-point values from xmm to m128, minimizing pollution in the cache hierarchy.
MOVNTQ	Move Quadword Non-Temporal	
PREFETCH0	Prefetch Data from Address	Prefetch into all cache levels
PREFETCH1	Prefetch Data from Address	Prefetch into all cache levels EXCEPT L1
PREFETCH2	Prefetch Data from Address	Prefetch into all cache levels EXCEPT L1 and L2
PREFETCHNTA	Prefetch Data from Address	Prefetch into all cache levels to non-temporal cache structure
SFENCE	Store Fence	Processor hint to make sure all store operatons after the SFENCE call are globally visible

Added with SSE2

Instruction	Meaning	Notes
CLFLUSH	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	Load Fence	Serializes load operations.
MASKMOVDQU	Masked Move of Double Quadword Unaligned	Stores selected bytes from the source operand (first operand) into a 128-bit memory location
MFENCE	Memory Fence	Performs a serializing operation on all load and store instructions that were issued prior the MFENCE instruction.
MOVNTDQ	Move Double Quadword Non-Temporal	Move double quadword from xmm to m128, minimizing pollution in the cache hierarchy.
MOVNTI	Move Doubleword Non-Temporal	Move doubleword from r32 to m32, minimizing pollution in the cache hierarchy.
MOVNTPD	Move Packed Double-Precision Floating-Point Values Non-Temporal	Move packed double-precision floating-point values from xmm to m128, minimizing pollution in the cache hierarchy.
PAUSE	Provides a hint to the processor that the following code is a spin loop	for Cacheability

Added with SSE3

Instruction	Meaning	Notes
LDDQU	Load Unaligned Integer 128 bits	Instructionally Equivalent to MOVDQU. Used for Video Encoding.
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 x86-64

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

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/From Control Registers	Move 32 or 64 Bit Contents to Control Register and Vise-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

VMPTRLD, VMPTRST, VMCLEAR, VMREAD, VMWRITE, VMCALL, VMLAUNCH, VMRESUME, VMXOFF, VMXON

Added with SSE4a

LZCNT, POPCNT (POPulation CouNT) - advanced bit manipulation

x87 floating-point instructions

Original 8087 instructions

Instruction	Meaning	Notes
F2XM1	$2^x - 1$	more precise than 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
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 $\log_2(e)$ onto stack	
FLDL2T	Load $\log_2(10)$ onto stack	
FLDLG2	Load $\log_{10}(2)$ 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	$y * \log_2 x$	if $y = \log_b 2$, then the base-b logarithm is computed
FYL2XP1	$y * \log_2 (x+1)$	more precise than $\log_2 z$ if x is close to zero

Added in specific processors

Added with 80287

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

Added with 80387

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

- FCMOV variants: FCMOVB, FCMOVBE, FCMOVE, FCMOVNB, FCMOVNBE, FCMOVNE, FCMOVNU, FCMOVU
- FCOMI variants: FCOMI, FCOMIP, FUCOMI, FUCOMIP

Added with SSE

FXRSTOR, FXSAVE

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

Added with SSE3

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

Undocumented x87 instructions

FFREEP performs FFREE ST(i) and pop stack

SIMD instructions

MMX instructions

Added with Pentium MMX

EMMS, MOVD, MOVQ, PACKSSDW, PACKSSWB, PACKUSWB, PADDB, PADDD, PADDSB, PADDSW, PADDUSB, PADDUSW, PADDW, PAND, PANDN, PCMPEQB, PCMPEQD, PCMPEQW, PCMPGTB, PCMPGTD, PCMPGTW, PMADDWD, PMULHW, PMULLW, POR, PSLLD, PSLLQ, PSLLW, PSRAD, PSRAW, PSRLD, PSRLQ, PSRLW, PSUBB, PSUBD, PSUBSB, PSUBSW, PSUBUSB, PSUBUSW, PSUBW, PUNPCKHBW, PUNPCKHDQ, PUNPCKHWD, PUNPCKLBW, PUNPCKLDQ, PUNPCKLWD, PXOR

MMX+ instructions

Added with Athlon

Same as the SSE SIMD integer instructions which operated on MMX registers.

EMMX instructions

EMMI instructions

(added with 6x86MX from Cyrix, deprecated now)

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

3DNow! instructions

Added with 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

Added with Athlon

PF2IW, PFNACC, PFPNACC, PI2FW, PSWAPD

Added with Geode GX

PFRSQRTV, PFRCPV

SSE instructions

Added with Pentium III

SSE SIMD floating-point instructions

ADDPS, ADDSS, CMPPS, CMPSS, COMISS, CVTPI2PS, CVTPS2PI, CVTSI2SS, CVTSS2SI, CVTTPS2PI, CVTTSS2SI, DIVPS, DIVSS, LDMXCSR, MAXPS, MAXSS, MINPS, MINSS, MOVAPS, MOVHLPS, MOVHPS, MOVLHPS, MOVLPS, MOVMSKPS, MOVNTPS, MOVSS, MOVUPS, MULPS, MULSS, RCPPS, RCPSS, RSQRTPS, RSQRTSS, SHUFPS, SQRTPS, SQRTSS, STMXCSR, SUBPS, SUBSS, UCOMISS, UNPCKHPS, UNPCKLPS

SSE SIMD integer instructions

ANDNPS, ANDPS, ORPS, PAVGB, PAVGW, PEXTRW, PINSRW, PMAXSW, PMAXUB, PMINSW, PMINUB, PMOVMSKB, PMULHUW, PSADBW, PSHUFW, XORPS

Instruction	Opcode	Meaning	Notes
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	
PREFETCHNTA	0F 18 /0	Prefetch Data Into Caches (non-temporal data with respect to all cache levels)	
PREFETCH0	0F 18 /1	Prefetch Data Into Caches (temporal data)	
PREFETCH1	0F 18 /2	Prefetch Data Into Caches (temporal data with respect to first level cache)	
PREFETCH2	0F 18 /3	Prefetch Data Into Caches (temporal data with respect to second level cache)	
NOP	0F 1F /0	No Operation	
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	
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	
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	

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	
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	
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	
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	
PSHUFW mm1, mm2/m64, imm8	0F 70 /r ib	Shuffle Packed Words	
LDMXCSR m32	0F AE /2	Load MXCSR Register State	
STMXCSR m32	0F AE /3	Store MXCSR Register State	
SFENCE	0F AE /7	Store Fence	
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	
PINSRW mm, r32/m16, imm8	0F C4 /r	Insert Word	
PEXTRW r32, mm, imm8	0F C5 /r	Extract Word	
SHUFPS xmm1, xmm2/m128, imm8	0F C6 /r ib	Shuffle Packed Single-Precision Floating-Point Values	
PMOVMSKB r32, 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	
MOVNTQ m64, mm	0F E7 /r	Store of Quadword Using Non-Temporal Hint	
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	
MASKMOVQ mm1, mm2	0F F7 /r	Store Selected Bytes of Quadword	

SSE2 instructions

Added with Pentium 4 Also see integer instructions added with Pentium 4

SSE2 SIMD floating-point instructions

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
ANDNPD xmm1, xmm2/m128	66 0F 55 /r	Bitwise Logical AND NOT
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

ADDPD, ADDSD, ANDNPD, ANDPD, CMPPD, CMPSD*, COMISD, CVTDQ2PD, CVTDQ2PS, CVTPD2DQ, CVTPD2PI, CVTPD2PS, CVTPI2PD, CVTPS2DQ, CVTPS2PD, CVTSD2SI, CVTSD2SS, CVTSI2SD, CVTSS2SD, CVTTPD2DQ, CVTTPD2PI, CVTTPS2DQ, CVTTSD2SI, DIVPD, DIVSD, MAXPD, MAXSD, MINPD, MINSD, MOVAPD, MOVHPD, MOVLPD, MOVMSKPD, MOVSD*, MOVUPD, MULPD, MULSD, ORPD, SHUFPD, SQRTPD, SQRTSD, SUBPD, SUBSD, UCOMISD, UNPCKHPD, UNPCKLPD, XORPD

- CMPSD and MOVSD have the same name as the string instruction mnemonics CMPSD (CMPS) and MOVSD (MOVS); however, the former refer to scalar double-precision floating-points whereas the latters refer to doubleword strings.

SSE2 SIMD integer instructions

MOVDQ2Q, MOVDQA, MOVDQU, MOVQ2DQ, PADDQ, PSUBQ, PMULUDQ, PSHUFHW, PSHUFLW, PSHUFD, PSLLDQ, PSRLDQ, PUNPCKHQDQ, PUNPCKLQDQ

SSE3 instructions

Added with Pentium 4 supporting SSE3 Also see integer and floating-point instructions added with Pentium 4 SSE3

SSE3 SIMD floating-point instructions

- ADDSUBPD, ADDSUBPS (for Complex Arithmetic)
- HADDPD, HADDPS, HSUBPD, HSUBPS (for Graphics)
- MOVDDUP, MOVSHDUP, MOVSLDUP (for Complex Arithmetic)

SSSE3 instructions

Added with Xeon 5100 series and initial Core 2

- PSIGNW, PSIGND, PSIGNB
- PSHUFB
- PMULHRSW, PMADDUBSW
- PHSUBW, PHSUBSW, PHSUBD
- PHADDW, PHADDSW, PHADDD
- PALIGNR
- PABSW, PABSD, PABSB

SSE4 instructions

SSE4.1

Added with Core 2 manufactured in 45nm

- MPSADBW
- PHMINPOSUW
- PMULLD, PMULDQ
- DPPS, DPPD
- BLENDPS, BLENDPD, BLENDVPS, BLENDVPD, PBLENDVB, PBLENDW
- PMINSB, PMAXSB, PMINUW, PMAXUW, PMINUD, PMAXUD, PMINSD, PMAXSD
- ROUNDPS, ROUNDSS, ROUNDPD, ROUNDSD
- INSERTPS, PINSRB, PINSRD/PINSRQ, EXTRACTPS, PEXTRB, PEXTRW, PEXTRD/PEXTRQ
- PMOVSXBW, PMOVZXBW, PMOVSXBD, PMOVZXBD, PMOVSXBQ, PMOVZXBQ, PMOVSXWD, PMOVZXWD, PMOVSXWQ, PMOVZXWQ, PMOVSXDQ, PMOVZXDQ
- PTEST
- PCMPEQQ
- PACKUSDW
- MOVNTDQA

SSE4a

Added with Phenom processors

- LZCNT, POPCNT (POPulation CouNT) - advanced bit manipulation
- EXTRQ/INSERTQ
- MOVNTSD/MOVNTSS

SSE4.2

Added with Nehalem processors

- CRC32
- PCMPESTRI
- PCMPESTRM
- PCMPISTRI
- PCMPISTRM
- PCMPGTQ

Intel AVX FMA instructions

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	

Intel AES instructions

Main article: AES instruction set

6 new instructions.

Instruction	Description
AESENC	Perform one round of an AES 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

Undocumented instructions

The x86 CPUs contain undocumented instructions 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 and at http://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.
UD1	0F B9	Intentionally undefined instruction, but unlike UD2 this was not published	
ICEBP	F1	Single byte single-step exception / Invoke ICE	Available beginning with 80386, documented (as INT1) since Pentium Pro
LOADALL	0F 05	Loads All Registers from Memory Address 0x000800H	Only available on 80286
Unknown mnemonic	0F 04	Exact purpose unknown, causes CPU hang. (the only way out is CPU reset)[2] In some implementations, emulated through BIOS as a halting sequence.[3]	Only available on 80286
LOADALLD	0F 07	Loads All Registers from Memory Address ES:EDI	Only available on 80386
POP CS	0F	Pop top of the stack into CS Segment register (causing a far jump)	Only available on earliest models of 8086. Beginning with 80286 this opcode is used as a prefix for 2-Byte-Instructions
MOV CS,r/m	8E/1	Moves a value from register/memory into CS Segment register (causing a far jump)	Only available on earliest models of 8086. Beginning with 80286 this opcode causes an invalid opcode exception
MOV ES,r/m	8E/4	Moves a value from register/memory into ES segment register	Only available on earliest models of 8086. On 80286 this opcode causes an invalid opcode exception. Beginning with 80386 the value is moved into the FS segment register.
MOV CS,r/m	8E/5	Pop top of the stack into CS Segment register (?)	Only available on earliest models of 8086. On 80286 this opcode causes an invalid opcode exception. Beginning with 80386 the value is moved into the GS segment register.
MOV SS,r/m	8E/6	Moves a value from register/memory into SS Segment register	Only available on earliest models of 8086. Beginning with 80286 this opcode causes an invalid opcode exception
MOV DS,r/m	8E/7	Moves a value from register/memory into DS Segment register	Only available on earliest models of 8086. Beginning with 80286 this opcode causes an invalid opcode exception

See also

- CLMUL
- XOP
- CVT16
- FMA
- RdRand
- Larrabee extensions
- Advanced Vector Extensions 2
- CPUID

References

1. ^ **a b** "Re: Intel® Processor Identification and the CPUID Instruction" (http://www.intel.com/content/www/us/en/processors/processor-identification-cpuid-instruction-note.html?wapkw=processor-identification-cpuid-instruction). Retrieved 2013-04-21.
2. ^ "Re: Undocumented opcodes (HINT_NOP)" (http://www.sandpile.org/post/msgs/20004129.htm). Retrieved 2010-11-07.
3. ^ "Re: Also some undocumented 0Fh opcodes" (http://www.sandpile.org/post/msgs/20003986.htm). Retrieved 2010-11-07.

- Intel Software Developer's Manuals (http://www.intel.com/products/processor/manuals/)

External links

- The 8086 / 80286 / 80386 / 80486 Instruction Set (http://web.archive.org/web/20100407092131/http://home.comcast.net/~fbui/intel.html)
- Free IA-32 and x86-64 documentation (http://www.intel.com/products/processor/manuals/index.htm), provided by Intel
- Netwide Assembler Instruction List (http://www.nasm.us/doc/nasmdocb.html) (from Netwide Assembler)
- x86 Opcode and Instruction Reference (http://ref.x86asm.net)