x86 Assembly

Suffixes

- **z** - zword; 512-bits
- **u** - upper-word; yword or zword
- **y**- yword; 256-bits
- **h** - half-word; qword, oword, or yword
- **o** - oword; 128-bits
- **f** - fourth-word; dword, qword, or oword
- **n** - normal-word; oword, yword, or zword
- **x** - xword; oword or yword
- **t** - ten-bytes; 80-bit float-point
- **q** - quadword; 64-bits
- **l** - longword/doubleword; 32-bit integer or 64-bit float-point
- **w** - word; 16-bits
- **e** - eighth-word; word, dword, or qword
- **s** - short; 16-bit integer or 32-bit float-point
- **b** - byte; 8-bits

Properties

- Little-endian
- Register-memory CISC
- Status register branching
- Last In First Out (LIFO) Stack

Ports

- **0x20-0x21** - Control/mask ports of the master PIC
- **0xa0-0xa1** - Control/mask ports of the slave PIC
- **0x60** - Data port from the keyboard controller
- **0x64** - Command port for keyboard controller (enable/disable kbd interrupts)

Datatypes

TYPE	63	62	31	30	15	14	7	6	4	3	0
Signed Byte							sign	integer			
Signed Word					sign	integer					
Signed Dword			sign	integer							
Signed Qword	sign	integer									
Unsigned Byte							integer				
Unsigned Word					integer						
Unsigned Dword			integer								
Unsigned Qword	integer										
BCD										BCD	
Packed BCD							BCD				
Word Integer					sign	integer					
Short Integer		sign	integer								
Long Integer	sign	integer									

TYPE	79	78	64	63	62	52	51	31	30	23	22	15	14	10	9	0
Half Real												sign	exp		fraction	
Single Real								sign	exp			fraction				
Double				sign	exp			fraction								
Extended Real	sign	exp		int	fraction											

TYPE	127	112	111	96	95	80	79	64	63	48	47	32	31	16	15	0
MMX Packed Word									word		word		word		word	
MMX Packed Dword									dword				dword			
MMX Qword									qword							
MMX Packed Word	word		word		word		word		word		word		word		word	
MMX Packed Dword	dword				dword				dword				dword			
MMX Qword	qword								qword							
SSE Scalar Single FP	reserved												FP			
SSE Packed Single FP	FP				FP				FP				FP			
SEE2 Scalar Double FP	reserved								double							
SSE2 Packed Double FP	double								double							
F16C Packed Half FP									Half FP		Half FP		Half FP		Half FP	
F16C Packed Half FP	Half FP		Half FP		Half FP		Half FP		Half FP		Half FP		Half FP		Half FP	

Created by Devyn Collier Johnson <DevynCJohnson@Gmail.com> (2016v2) More cheatsheets at DCJTech.info

x86 Assembly

TYPE	512	480	479	448	447	416	415	384	383	352	351	320	319	288	287	256	255	224	223	192	191	160	159	128	127	96	95	64	63	32	31	0
AVX Packed Dword																	dword		dword		dword		dword		dword		dword		dword		dword	
AVX Packed Qword																	qword				qword				qword				qword			
AVX Scalar Single FP																	reserved														FP	
AVX Scalar Double FP																	reserved												double			
AVX Packed Single FP																	FP		FP		FP		FP		FP		FP		FP		FP	
AVX Packed Double FP																	double				double				double				double			
MVEX Scalar Single FP	reserved																														FP	
MVEX Scalar Double FP	reserved																												double			
MVEX Packed Single FP	FP		FP		FP		FP		FP		FP		FP		FP		FP		FP		FP		FP		FP		FP		FP		FP	
MVEX Packed Double FP	double				double				double				double				double				double				double				double			

NOTE: MVEX and EVEX datatypes are the same

Condition Codes

Code	Bit 3	Bit 2	Bit 1	Bit 0	Condition
O	0	0	0	0	overflow
NO	0	0	0	1	no overflow
B (NAE, C)	0	0	1	0	below (not above or equal, carry)
NB (AE, NC)	0	0	1	1	not below (above or equal, no carry)
E (Z)	0	1	0	0	equal (zero)
NE (NZ)	0	1	0	1	not equal (not zero)
NA (BE)	0	1	1	0	not above (below or equal)
A (NBE)	0	1	1	1	above (not below or equal)
S	1	0	0	0	sign
NS	1	0	0	1	no sign
P (PE)	1	0	1	0	parity (parity even)
NP (PO)	1	0	1	1	no parity (parity odd)
L (NGE)	1	1	0	0	less than (not greater than or equal)
NL (GE)	1	1	0	1	not less than (greater than or equal)
NG (LE)	1	1	1	0	not greater than (less than or equal)
G (NLE)	1	1	1	1	greater than (not less than or equal)

Registers

Bounds Registers
128-bit registers;
64 MSBs for upper-bound;
64 LSBs for lower-bound; BND0, BND1, BND2, BND3

Config and Status Registers
- **BNDCFGS** - EN (bit 0), BPRV (bit 1), BD_BASE (bits 12-63)
- **BNDCFGU** - EN (bit 0), BPRV (bit 1), BD_BASE (bits 12-63)
- **BNDSTATUS** - EC (bits 0-1), BDE_ADDRESS (bits 2-63)

Control Registers
Control registers change or control the general behavior of the CPU, co-processor, or other digital device. Behaviors include interrupts, addressing mode, paging, and more.

- **CR0**
- **0 (PE)** - Protected Mode Enabled; If 1, then system is in protected mode, else the system is in real mode
- **1 (MP)** - Monitor co-processor; Controls interaction of WAIT/FWAIT instructions with TS flag in CR0
- **2 (EM)** - Emulation; If set, no x87 floating point unit present, if clear, x87 FPU present
- **3 (TS)** - Task Switched; Allows saving x87 task context upon a task switch only after x87 instruction used
- **4 (ET)** - Extension Type; On the 386, specify whether the external math co-processor IS an 80287 or 80387
- **5 (NE)** - Numeric Error; Enable internal x87 floating point error reporting when set, else enables PC style x87 error detection
- **16 (WP)** - Write Protect; When set, the CPU can not write to read-only pages when privilege level is 0
- **18 (AM)** - Alignment mask; Alignment check enabled if AM set, AC flag (in EFLAGS register) set, and privilege level is 3
- **29 (NW)** - Not-write through; Globally enables/disable write-through caching
- **30 (CD)** - Cache Disable; Globally enables/disable the memory cache
- **31 (PG)** - Paging; If 1, enable paging and use the CR3 register, else disable paging

Created by Devyn Collier Johnson <DevynCJohnson@Gmail.com> (2016v2) More cheatsheets at DCJTech.info

x86 Assembly

- **CR1** - Reserved
- **CR2** - Page Fault Linear Address (PFLA); When a page fault occurs, the address the program attempted to access is stored in the CR2 register.
- **CR3 (PDBR)** - Page Directory Base Register (Upper 20 bits). Used when virtual addressing is enabled to translate linear addresses into physical addresses by locating the page directory and page tables for the current task
- **CR4** - Used in protected mode to control operations such as virtual-8086 support, enabling I/O breakpoints, page size extension and machine check exceptions.
- **0 (VME)** - Virtual 8086 Mode Extensions; If set, enables support for the virtual interrupt flag (VIF) in virtual-8086 mode
- **1 (PVI)** - Protected-mode Virtual Interrupts; If set, enables support for the virtual interrupt flag (VIF) in protected mode
- **2 (TSD)** - Time Stamp Disable; If set, RDTSC instruction can only be executed when in ring 0, otherwise RDTSC can be used at any privilege level
- **3 (DE)** - Debugging Extensions; If set, enables debug register based breaks on I/O space access
- **4 (PSE)** - Page Size Extension; If unset, page size is 4 KiB, else page size is increased to 4 MiB (or 2 MiB with PAE set)
- **5 (PAE)** - Physical Address Extension; If set, changes page table layout to translate 32-bit virtual addresses into extended 36-bit physical addresses
- **6 (MCE)** - Machine Check Exception; If set, enables machine check interrupts to occur
- **7 (PGE)** - Page Global Enabled; If set, address translations (PDE or PTE records) may be shared between address spaces
- **8 (PCE)** - Performance-Monitoring Counter enable; If set, RDPMC can be executed at any privilege level, else RDPMC can only be used in ring 0
- **9 (OSFXSR)** - Operating system support for FXSAVE and FXRSTOR instructions; If set, enables SSE instructions and fast FPU save & restore
- **10 (OSXMMEXCPT)** - Operating System Support for Unmasked SIMD Floating-Point Exceptions; If set, enables unmasked SSE exceptions
- **13 (VMXE)** - Virtual Machine Extensions Enable
- **14 (SMXE)** - Safer Mode Extensions Enable; Trusted Execution Technology (TXT)
- **16 (FSGSBASE)** - Enables the instructions RDFSBASE, RDGSBASE, WRFSBASE, and WRGSBASE
- **17 (PCIDE)** - PCID Enable; If set, enables process-context identifiers (PCIDs)
- **18 (OSXSAVE)** - XSAVE and Processor Extended States Enable
- **20 (SMEP)** - Supervisor Mode Execution Protection Enable; If set, execution of code in a higher ring generates a fault
- **21 (SMAP)** - Supervisor Mode Access Protection Enable; If set, access of data in a higher ring generates a fault
- **22 (PKE)** - Protection Key Enable
- **EFER (MSR 0xC0000080)** - Extended Feature Enable Register; x86-64 only
- **0 (SCE)** - System Call Extensions
- **8 (LME)** - Long Mode Enable
- **10 (LMA)** - Long Mode Active
- **11 (NXE)** - No-Execute Enable
- **12 (SVME)** - Secure Virtual Machine Enable
- **13 (LMSLE)** - Long Mode Segment Limit Enable
- **14 (FFXSR)** - Fast FXSAVE/FXRSTOR
- **15 (TCE)** - Translation Cache Extension
- **CR5-CR7** - Reserved
- **CR8 (TPR)** - Task-Priority Register; Prioritize external interrupts; x86-64 only
- **CR9-CR15** - Reserved

Debug Registers

Used for program debugging; only accessible at Privilege Level 0

- **DR0-DR3** - Contains a linear address associated with one of four breakpoint conditions; Each breakpoint condition is further defined by bits in DR7
- **DR4** - Alias for DR6
- **DR5** - Alias for DR7
- **DR6** - The debug status register permits the debugger to determine which debug conditions have occurred. When the processor detects an enabled debug exception, it sets the low-order bits of this register (0,1,2,3) before entering the debug exception handler. Move zeros into this register after use because this register never clears itself
- **DR7** - The low-order eight bits of DR7 (0,2,4,6 and 1,3,5,7) selectively enable the four address breakpoint conditions. There are two levels of enabling: the local (0,2,4,6) and global (1,3,5,7) levels. The local enable bits are automatically reset by the processor at every task switch to avoid unwanted breakpoint conditions in the new task. The global enable bits are not reset by a task switch; therefore, they can be used for conditions that are global to all tasks. Bits 16-17 (DR0), 20-21 (DR1), 24-25 (DR2), 28-29 (DR3), define when breakpoints trigger. Each breakpoint has a two-bit entry that specifies whether they break on execution (00b), data write (01b), data read or write (11b). 10b is defined to mean break on IO read or write but no hardware supports it; for 64-bit mode, it has been re-purposed to specify an 8 byte wide breakpoint area. Bits 18-19 (DR0), 22-23 (DR1), 26-27 (DR2), 30-31 (DR3), define how large an area of memory is watched by breakpoints. Again each breakpoint has a two-bit entry that specifies whether they watch one (00b), two (01b), eight (10b) or four (11b) bytes.
- **DR8-DR15** - Reserved

Fixed Range MTRRs (64-bits)

- **MTRR_FIX_64K_00000**
- **MTRR_FIX_16K_80000**
- **MTRR_FIX_16K_A0000**
- **MTRR_FIX_4K_C0000**
- **MTRR_FIX_4K_C8000**
- **MTRR_FIX_4K_D0000**
- **MTRR_FIX_4K_D8000**
- **MTRR_FIX_4K_E0000**
- **MTRR_FIX_4K_E8000**
- **MTRR_FIX_4K_F0000**
- **MTRR_FIX_4K_F8000**

Machine Check Architecture (64-bits)

- **MCG_CAP** -
- **MCG_STATUS** - RIPV (bit 0), EIPV (bit 1), MCIP (bit 2), LMCES (bit 3)
- **MCG_CTL** - reserved
- **MCG_EXT_CTL** - LMCEEN (bit 0)

Machine Check Exception

- **MCAR** - 32-bits
- **MCTR** - 32-bits; CHK (0), WR (1), DC (2), MIO (3), LCK (4)

Created by Devyn Collier Johnson <DevynCJohnson@Gmail.com> (2016v2) More cheatsheets at DCJTech.info

x86 Assembly

FLAGS, EFLAGS, & RFLAGS Registers

- **0 (CF)** - Carry Flag. Set if the last arithmetic operation carried (addition) or borrowed (subtraction) a bit beyond the size of the register. This is then checked when the operation is followed with an add-with-carry or subtract-with-borrow to deal with values too large for just one register to contain
- **2 (PF)** - Parity Flag. Set if the number of set bits in the least significant byte is a multiple of 2
- **4 (AF)** - Adjust Flag. Carry of Binary Code Decimal (BCD) numbers arithmetic operations
- **6 (ZF)** - Zero Flag. Set if the result of an operation is Zero (0)
- **7 (SF)** - Sign Flag. Set if the result of an operation is negative
- **8 (TF)** - Trap Flag. Set if step by step debugging
- **9 (IF)** - Interruption Flag. Set if interrupts are enabled
- **10 (DF)** - Direction Flag. Stream direction. If set, string operations will decrement their pointer rather than incrementing it, reading memory backwards
- **11 (OF)** - Overflow Flag. Set if signed arithmetic operations result in a value too large for the register to contain
- **12-13 (IOPL)** - I/O Privilege Level field. I/O Privilege Level of the current process
- **14 (NT)** - Nested Task flag. Controls chaining of interrupts. Set if the current process is linked to the next process
- **16 (RF)** - Resume Flag. Response to debug exceptions
- **17 (VM)** - Virtual-8086 Mode. Set if in 8086 compatibility mode
- **18 (AC)** - Alignment Check. Set if alignment checking of memory references is done
- **19 (VIF)** - Virtual Interrupt Flag. Virtual image of IF
- **20 (VIP)** - Virtual Interrupt Pending flag. Set if an interrupt is pending
- **21 (ID)** - Identification Flag. Support for CPUID instruction if can be set
- **22-63** - NULL/VOID/Unassigned

General Purpose Registers

- **AL/AH/AX/EAX/RAX** - Accumulator (add immediates and perform math)
- **BL/BH/BX/EBX/RBX** - Base index (for use with arrays)
- **CL/CH/CX/ECX/RCX** - Counter (for use with loops and strings)
- **DL/DH/DX/EDX/RDX** - Extend the precision of the accumulator (combine EAX and EDX for 64-bit integer operations)
- **BP/EBP/RBP** - Stack base pointer for holding the address of the current stack frame
- **DI/EDI/RDI** - Destination index for string operations
- **IP/EIP/RIP** - Instruction pointer (holds the program counter and the current instruction address)
- **R8-R15** - General 64-bit (quadword) registers
- **SI/ESI/RSI** - Source index for string operations
- **SP/ESP/RSP** - Stack pointer for top address of the stack

MCA Error-Reporting Register Banks (64-bits)

- **MCn_CTL2** - Intel P6-core processors alias MC0_CTL to EBL_CR_POWERON
- **MCn_STATUS** -
- **MCn_ADDR** -
- **MCn_MISC - reserved**

Memory Type Range Registers (MTRR)

- **MTRR_CAP** - 64-bits
- **0-7 (VCNT)** -
- **8 (FIX)** -
- **10 (WC)** -
- **11 (SMRR)** -
- **MTRR_DEF_TYPE** - 64-bits
- **0-7 (TYPE)** -
- **10 (FE)** -
- **11 (E)** -

SMRRs

64-bit registers with the most-significant 32-bits reserved

- **SMRR_PHYS_BASE** - TYPE (bits 0-7); BASE (bits 12-31)
- **SMRR_PHYS_MASK** - V (bit 11); MASK (bits 12-31)

Segment Registers

Segment registers are 128-bits. However, only the MSB 16 bits can be used (called the segment selector). The rest are used as an "internal descriptor cache". 16 bits of this cache is used for "access rights", 32 bits for "limit", and either 32 or 64 bits for "base".

- **Stack Segment (SS)** - Pointer to the stack
- **Code Segment (CS)** - Pointer to the code
- **Data Segment (DS)** - Pointer to the data
- **Extra Segment (ES)** - Pointer to extra data
- **F Segment (FS)** - Pointer to extra data #2
- **G Segment (GS)** - Pointer to extra data #3

Table Registers

Defines memory segments and ensures protection mode operations.

- **GDTR** - Global Descriptor Table Register; Stores the size and memory location of the GDT
- **IDTR** - Interrupt Descriptor Table Register; Interrupt vector table; The IDT is used by the processor to determine the correct response to interrupts and exceptions
- **LDTR** - Local Descriptor Table Register; Stores the characteristics of the local memory segments
- **TR** - Table Register

MCA Extended State Registers

64-bit registers; MCG_rAX, MCG_rBX, MCG_rCX, MCG_rDX, MCG_rSI, MCG_rDI, MCG_rBP, MCG_rSP, MCG_rFLAGS, MCG_rIP, MCG_MISC, MCG_RESx (128-bit), MCG_R8-MCG_R15.

Page Attribute Table (PAT)

- **0-2 (PA0)** -
- **3-7 (reserved)** -
- **8-10 (PA1)** -
- **11-15 (reserved)** -
- **16-18 (PA2)** -
- **19-23 (reserved)** -
- **24-26 (PA3)** -
- **27-31 (reserved)** -
- **32-34 (PA4)** -
- **35-39 (reserved)** -
- **40-42 (PA5)** -
- **43-47 (reserved)** -
- **48-50 (PA6)** -
- **51-55 (reserved)** -
- **56-58 (PA7)** -
- **59-63 (reserved)** -

Test Registers

Used to perform self-tests; 80486 only;

- **TR0-TR5** - Undocumented Test Registers
- **TR6** - Test commands
- **TR7** - Test data

Time Stamp Counter (MSR)

- **TSC** - Time Stamp Counter
- **TSC_ADJUST** - Adjust TSC
- **TSC_AUX** - Processor ID (32-bits)
- **MPERF** - Maximum clock frequency
- **APERF** - Actual clock frequency

Variable Range MTRRs

64-bit registers

- **MTRR_PHYS_BASE_n**
- **MTRR_PHYS_MASK_n**

Created by Devyn Collier Johnson <DevynCJohnson@Gmail.com> (2016v2) More cheatsheets at DCJTech.info

x86 Assembly

512-bits	256-bits	128-bits	79-bits	78-bits	64-bits (qword)	32-bits (longword)	16-bits (word)	8-bits (LSB)
General Purpose Registers (GPR)								
Accumulator								
					RAX || R0	EAX || R0D	AH	AL
								AX
Base Index								
					RBX || R3	EBX || R3D	BH	BL
								BX
Counter Index								
					RCX || R1	ECX || R1D	CH	CL
								CX
Data Register								
					RDX || R2	EDX || R2D	DH	DL
								DX
Destination Index								
					RDI || R7	EDI || R7D	DI	
Instruction Pointer								
					RIP	EIP	IP	
Registers (R8-R15)								
					R*	R*D	R*W	R*B
Source Index								
					RSI || R6	ESI || R6D	SI	
Stack Base Pointer								
					RBP || R5	EBP || R5D	BP	BPL (64-bit mode only)
Stack Pointer								
					RSP || R4	ESP || R4D	SP	SPL (64-bit mode only)
Segment Registers (C, D, E, F, G, S)								
								*S
Mask Registers								
					K0-K7			
Debug Registers								
					DR0-DR15			
Control Registers								
						MSW		
					CR0			
					CR1-CR2			
						CR3 (32-bit)		
					CR3 (64-bit)			
					CR4			
					CR5-CR7			
					CR8			TPR (4-bits)
					CR9-CR15			
Model Specific Registers (MSRs)								
Time Stamp Counter								
					TSC			
					TSC_ADJUST			
					reserved	TSC_AUX		

Created by Devyn Collier Johnson <DevynCJohnson@Gmail.com> (2016v2) More cheatsheets at DCJTech.info

x86 Assembly

512-bits	256-bits	128-bits	79-bits	78-bits	64-bits (qword)	32-bits (longword)	16-bits (word)	8-bits (LSB)
					MPERF			
					APERF			
SMM-Related Internal Registers								
					SMBASE			
					IORESTARTRIP			
					IORESTARTRCX			
					IORESTARTRSI			
					IORESTARTRDI			
Miscellaneous MSRs								
						MISC_CTL		
					MISC_ENABLE			
						EFER		
					ignored	scratch	SEP_SEL	
					ignored	SEP_ESP		
					SEP_RSP			
					ignored	SEP_EIP		
					SEP_RIP			
					STAR			
					LSTAR			
					CSTAR			
					reserved	FMASK		
					FS_BAS			
					GS_BAS			
					KERNEL_GS_BAS			
					APIC_BASE			
					BNDCFGS			
					XSS			
FP/MMX/3DNow! Registers								
			sign	exp	ST0-ST7 || MM0-MM7 || MMX0-MMX7 || FPR0-FPR7			
SSE Vector FP Registers								
		XMM0-XMM7						
	YMM0-YMM7							
ZMM0-ZMM7								
SSE2 Vector FP Registers								
		XMM8-XMM15						
	YMM8-YMM15							
ZMM8-ZMM15								
AVX Vector FP Registers (MVEX)								
ZMM16-ZMM31								
EVEX Vector FP Registers								
		XMM16-XMM31						
	YMM16-YMM31							
ZMM16-ZMM31								
512-bits	**256-bits**	**128-bits**	**79-bits**	**78-bits**	**64-bits (qword)**	**32-bits (longword)**	**16-bits (word)**	**8-bits (LSB)**

x86 Assembly

Branching and Conditionals

Mnemonic	Description
BOUND	Array index in source register is checked against upper and lower bounds in memory source. The first word located at "limit" is the lower boundary and the word at "limit+2" is the upper array bound. Interrupt 5 occurs if the source value is less than or higher than the source.
BT	The destination bit indexed by the source value is copied into the Carry Flag.
BTC	The destination bit indexed by the source value is copied into the Carry Flag after being complimented (inverted).
BTR	The destination bit indexed by the source value is copied into the Carry Flag and then cleared in the destination.
BTS	The destination bit indexed by the source value is copied into the Carry Flag and then set in the destination.
CALL	Pushes Instruction Pointer (and Code Segment for far calls) onto stack and loads Instruction Pointer with the address of the procedure.
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
CMP	Subtracts source from destination and updates the flags but does not save result. Flags can subsequently be checked for conditions. Modified: AF, CF, OF, PF, SF, ZF
CMPS	Subtracts destination value from source without saving results. Updates flags based on the subtraction and the index registers (E)SI and (E)DI are incremented or decremented depending on the state of the Direction Flag. CMPSB inc/decrements the index registers by 1, CMPSW inc/decrements by 2, while CMPSD increments or decrements by 4. Modified: AF, CF, OF, PF, SF, ZF
CMPSQ	CoMPare String Quadword
CMPXCHG	Compares the accumulator with "dest". If equal the "dest" is loaded with "src", otherwise the accumulator is loaded with "dest". Modified: AF, CF, OF, PF, SF, ZF
CMPXCHG16B	CoMPare and eXCHanGe 16 Bytes
ENTER	Modifies stack for entry to procedure for high level language. Operand "locals" specifies the amount of storage to be allocated on the stack. "Level" specifies the nesting level of the routine. Paired with the LEAVE instruction, this is an efficient method of entry and exit to procedures.
ESC	Provides access to the data bus for other resident processors. The CPU treats it as a NOP but places memory operand on bus.
HLT	Halts CPU until RESET line is activated, NMI or maskable interrupt received. The CPU becomes dormant but retains the current CS:IP for later restart.
ICEBP	ICE BreakPoint; An undocumented op code that will make debugging run-time code on an ICE easier
INT	Initiates a software interrupt by pushing the flags, clearing the Trap and Interrupt Flags, pushing CS followed by IP and loading CS:IP with the value found in the interrupt vector table. Execution then begins at the location addressed by the new CS:IP. Modified: IF, TF
INTO	If the Overflow Flag is set this instruction generates an INT 4 which causes the code addressed by 0000:0010 to be executed. Modified: IF, TF
IRET	Returns control to point of interruption by popping IP, CS and then the Flags from the stack and continues execution at this location. CPU exception interrupts will return to the instruction that cause the exception because the CS:IP placed on the stack during the interrupt is the address of the offending instruction. Modified: AF, CF, DF, IF, PF, SF, TF, ZF
IRETQ	64-bit Return from Interrupt
JA	Jump if Above; CF=0 and ZF=0
JAE	Jump if Above or Equal; CF=0
JB	Jump if Below; CF=1
JBE	Jump if Below or Equal; CF=1 or ZF=1
JC	Jump if Carry; CF=1
JCXZ	Jump if CX Zero;CX=0
JE	Jump if Equal; ZF=1
JG	Jump if Greater (signed); ZF=0 and SF=OF
JGE	Jump if Greater or Equal (signed); SF=OF
JL	Jump if Less (signed); SF != OF
JLE	Jump if Less or Equal (signed); ZF=1 or SF != OF
JMP	Unconditional Jump
JNA	Jump if Not Above; CF=1 or ZF=1
JNAE	Jump if Not Above or Equal; CF=1
JNB	Jump if Not Below; CF=0
JNBE	Jump if Not Below or Equal; CF=0 and ZF=0
JNC	Jump if Not Carry; CF=0
JNE	Jump if Not Equal; ZF=0
JNG	Jump if Not Greater (signed); ZF=1 or SF != OF
JNGE	Jump if Not Greater or Equal (signed); SF != OF
JNL	Jump if Not Less (signed); SF=OF

Created by Devyn Collier Johnson <DevynCJohnson@Gmail.com> (2016v2) More cheatsheets at DCJTech.info

x86 Assembly

JNLE	Jump if Not Less or Equal (signed); ZF=0 and SF=OF
JNO	Jump if Not Overflow (signed); OF=0
JNP	Jump if No Parity; PF=0
JNS	Jump if Not Signed (signed); SF=0
JNZ	Jump if Not Zero; ZF=0
JO	Jump if Overflow (signed); OF=1
JP	Jump if Parity; PF=1
JPE	Jump if Parity Even; PF=1
JPO	Jump if Parity Odd; PF=0
JRCXZ	Jump if RCX is zero
JS	Jump if Signed (signed); SF=1
JZ	Jump if Zero; ZF=1
LEAVE	Releases the local variables created by the previous ENTER instruction by restoring SP and BP to their condition before the procedure stack frame was initialized.
LOCK	This instruction is a prefix that causes the CPU assert bus lock signal during the execution of the next instruction. Used to avoid two processors from updating the same data location. The 286 always asserts lock during an XCHG with memory operands. This should only be used to lock the bus prior to XCHG, MOV, IN and OUT instructions.
LOOP	Decrements CX by 1 and transfers control to "label" if CX is not Zero. The "label" operand must be within -128 or 127 bytes of the instruction following the loop instruction.
LOOPE/LOOPZ	Decrements CX by 1 (without modifying the flags) and transfers control to "label" if CX != 0 and the Zero Flag is set. The "label" operand must be within -128 or 127 bytes of the instruction following the loop instruction.
LOOPNZ/LOOPNE	Decrements CX by 1 (without modifying the flags) and transfers control to "label" if CX != 0 and the Zero Flag is clear. The "label" operand must be within -128 or 127 bytes of the instruction following the loop instruction.
MONITOR	Setup Monitor Address; Sets up a linear address range to be monitored by hardware and activates the monitor. SSE3
MWAIT	Monitor Wait; Hint to stop instruction execution and enter an implementation-dependent optimized state until occurrence of a class of events. SSE3
NOP	Do nothing
PAUSE	Provides a hint to the processor that the following code is a spin loop; used for cacheability
REP	Repeats execution of string instructions while CX != 0. After each string operation, CX is decremented and the Zero Flag is tested.
REPE/REPZ	Repeats execution of string instructions while CX != 0 and the Zero Flag is set. CX is decremented and the Zero Flag tested after each string operation.
REPNE/REPNZ	Repeats execution of string instructions while CX != 0 and the Zero Flag is clear. CX is decremented and the Zero Flag tested after each string operation.
RET/RETF/RETN	Transfers control from a procedure back to the instruction address saved on the stack. "num_bytes" is an optional number of bytes to release. Far returns pop the IP followed by the CS, while near returns pop only the IP register.
RSM	This was introduced by the i386SL and later and is also in the i486SL and later. Resumes from System Management Mode (SMM).
SCAS	Compares value at ES:DI (even if operand is specified) from the accumulator and sets the flags similar to a subtraction. DI is incremented/decremented based on the instruction format (or operand size) and the state of the Direction Flag. Modified: AF, CF, OF, PF, SF, ZF
SETAE/SETNB	Sets the byte in the operand to 1 if the Carry Flag is clear otherwise sets the operand to 0.
SETB/SETNAE	Sets the byte in the operand to 1 if the Carry Flag is set otherwise sets the operand to 0.
SETBE/SETNA	Sets the byte in the operand to 1 if the Carry Flag or the Zero Flag is set, otherwise sets the operand to 0.
SETE/SETZ	Sets the byte in the operand to 1 if the Zero Flag is set, otherwise sets the operand to 0.
SETNE/SETNZ	Sets the byte in the operand to 1 if the Zero Flag is clear, otherwise sets the operand to 0.
SETL/SETNGE	Set if Less / Set if Not Greater or Equal
SETGE/SETNL	Sets the byte in the operand to 1 if the Sign Flag equals the Overflow Flag, otherwise sets the operand to 0.
SETLE/SETNG	Sets the byte in the operand to 1 if the Zero Flag is set or the Sign Flag is not equal to the Overflow Flag, otherwise sets the operand to 0.
SETG/SETNLE	Sets the byte in the operand to 1 if the Zero Flag is clear or the Sign Flag equals to the Overflow Flag, otherwise sets the operand to 0.
SETS	Sets the byte in the operand to 1 if the Sign Flag is set, otherwise sets the operand to 0.
SETNS	Sets the byte in the operand to 1 if the Sign Flag is clear, otherwise sets the operand to 0.
SETC	Sets the byte in the operand to 1 if the Carry Flag is set, otherwise sets the operand to 0.
SETNC	Sets the byte in the operand to 1 if the Carry Flag is clear, otherwise sets the operand to 0.
SETO	Sets the byte in the operand to 1 if the Overflow Flag is set, otherwise sets the operand to 0.
SETNO	Sets the byte in the operand to 1 if the Overflow Flag is clear, otherwise sets the operand to 0.
SETP/SETPE	Sets the byte in the operand to 1 if the Parity Flag is set, otherwise sets the operand to 0.
SETNP/SETPO	Sets the byte in the operand to 1 if the Parity Flag is clear, otherwise sets the operand to 0.

Created by Devyn Collier Johnson <DevynCJohnson@Gmail.com> (2016v2) More cheatsheets at DCJTech.info

x86 Assembly

STI	Sets the Interrupt Flag to 1, which enables recognition of all hardware interrupts. If an interrupt is generated by a hardware device, an End of Interrupt (EOI) must also be issued to enable other hardware interrupts of the same or lower priority.
TEST	Performs a logical AND of the two operands updating the flags register without saving the result. Modified: AF, CF, OF, PF, SF, ZF
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.
VERR	Verifies the specified segment selector is valid and is readable at the current privilege level. If the segment is readable, the Zero Flag is set, otherwise it is cleared.
VERW	Verifies the specified segment selector is valid and is writable at the current privilege level. If the segment is writable, the Zero Flag is set, otherwise it is cleared.
WAIT/FWAIT	CPU enters wait state until the coprocessor signals it has finished its operation. This instruction is used to prevent the CPU from accessing memory that may be temporarily in use by the coprocessor. WAIT and FWAIT are identical.
WBINVD	Flushes internal cache, then signals the external cache to write back current data followed by a signal to flush the external cache.

Data Manipulation

Mnemonic	Description
AAA	ASCII adjust AL after addition; used when unpacked binary coded decimal; Modified: AF, CF, OF, PF, SF, ZF
AAD	ASCII adjust AX before division; Modified: AF, CF, OF, PF, SF, ZF
AAM	ASCII adjust AX after multiplication Modified: AF, CF, OF, PF, SF, ZF;
AAS	ASCII adjust AL after subtraction; Modified: AF, CF, OF, PF, SF, ZF
ADC	Add with carry; Modified: AF, CF, OF, PF, SF, ZF
ADD	Add; Modified: AF, CF, OF, PF, SF, ZF
AND	Logical AND; Modified: AF, CF, OF, PF, SF, ZF
ARPL	Adjusted Requested Privilege Level of Selector; Compares the RPL bits of "dest" against "src". If the RPL bits of "dest" are less than "src", the destination RPL bits are set equal to the source RPL bits and the Zero Flag is set. Otherwise the Zero Flag is cleared.
BSF	Scans source operand for first bit set. Sets ZF if a bit is found set and loads the destination with an index to first set bit. Clears ZF if no bits are found set.
BSR	Scans source operand for first bit set. Sets ZF if a bit is found set and loads the destination with an index to first set bit. Clears ZF if no bits are found set.
BSWAP	Changes the byte order of a 32 bit register from big endian to little endian or vice versa. Result left in destination register is undefined if the operand is a 16 bit register.
CBW	Convert byte in AL to a word in AX
CDQ	Converts a signed dword in EAX to a signed quadword in EDX:EAX
CDQE	Sign extend EAX into RAX
CLC	Clear the carry bit
CLD	Clear the direction flag
CLI	Disables the maskable hardware interrupts by clearing the Interrupt flag. NMI's and software interrupts are not inhibited.
CLTS	Clears the Task Switched Flag in the Machine Status Register. This is a privileged operation and is generally used only by operating system code. Modified: IF
CMC	Toggle the carry flag; Modified: CF
CQO	Sign extend RAX into RDX:RAX
CWD	Extends sign of word in register AX throughout register DX forming a doubleword quantity in DX:AX.
CWDE	Converts a signed word in AX to a signed doubleword in EAX by extending the sign bit of AX throughout EAX.
DAA	Corrects result (in AL) of a previous BCD addition operation. Contents of AL are changed to a pair of packed decimal digits. Modified: AF, CF, OF, PF, SF, ZF
DAS	Corrects result (in AL) of a previous BCD subtraction operation. Contents of AL are changed to a pair of packed decimal digits. Modified: AF, CF, OF, PF, SF, ZF
DEC	Decrement; Modified: AF, OF, PF, SF, ZF
DIV	Unsigned binary division; Modified: AF, CF, OF, PF, SF, ZF
IDIV	Signed binary division; Modified: AF, CF, OF, PF, SF, ZF
IMUL	Signed multiplication; Modified: AF, CF, OF, PF, SF, ZF
IN	A byte, word or dword is read from "port" and placed in AL, AX or EAX respectively. If the port number is in the range of 0-255 it can be specified as an immediate, otherwise the port number must be specified in DX. Valid port ranges on the PC are 0-1024, though values through 65535 may be specified and recognized by third party vendors and PS/2's.
INC	Increment; Modified: AF, OF, PF, SF, ZF
INS	Loads data from port to the destination ES:(E)DI (even if a destination operand is supplied). (E)DI is adjusted by the size of the operand and increased if the Direction Flag is cleared and decreased if the Direction Flag is set. For INSB, INSW, INSD no operands are allowed and the size is determined by the mnemonic.

Created by Devyn Collier Johnson <DevynCJohnson@Gmail.com> (2016v2) More cheatsheets at DCJTech.info

x86 Assembly

INVD	Flushes CPU internal cache. Issues special function bus cycle which indicates to flush external caches. Data in write-back external caches is lost.
INVLPG	Invalidates a single page table entry in the Translation Look-Aside Buffer.
MUL	Unsigned multiply of the accumulator by the source. If "src" is a byte value, then AL is used as the other multiplicand and the result is placed in AX. If "src" is a word value, then AX is multiplied by "src" and DX:AX receives the result. If "src" is a double word value, then EAX is multiplied by "src" and EDX:EAX receives the result. Modified: AF, CF, OF, PF, SF, ZF
NEG	Subtracts the destination from 0 and saves the 2s complement of "dest" back into "dest". Modified: AF, CF, OF, PF, SF, ZF
NOT	Inverts the bits of the "dest" operand forming the 1s complement.
OR	Logical inclusive OR of the two operands returning the result in the destination. Modified: AF, CF, OF, PF, SF, ZF
RCL	Rotates the bits in the destination to the left "count" times with all data pushed out the left side re-entering on the right. The Carry Flag holds the last bit rotated out. Modified: CF, OF
RCR	Rotates the bits in the destination to the right "count" times with all data pushed out the right side re-entering on the left. The Carry Flag holds the last bit rotated out. Modified: CF, OF
ROL	Rotates the bits in the destination to the left "count" times with all data pushed out the left side re-entering on the right. The Carry Flag will contain the value of the last bit rotated out. Modified: CF, OF
ROR	Rotates the bits in the destination to the right "count" times with all data pushed out the right side re-entering on the left. The Carry Flag will contain the value of the last bit rotated out. Modified: CF, OF
SAL/SHL	Shifts the destination left by "count" bits with zeros shifted in on right. The Carry Flag contains the last bit shifted out. Modified: AF, CF, OF, PF, SF, ZF
SAR	Shifts the destination right by "count" bits with the current sign bit replicated in the leftmost bit. The Carry Flag contains the last bit shifted out. Modified: AF, CF, OF, PF, SF, ZF
SBB	Subtracts the source from the destination, and subtracts 1 extra if the Carry Flag is set. Results are returned in "dest". Modified: AF, CF, OF, PF, SF, ZF
SCASQ	SCAn String Quadword
SHR	Shifts the destination right by "count" bits with zeros shifted in on the left. The Carry Flag contains the last bit shifted out. Modified: AF, CF, OF, PF, SF, ZF
SHLD/SHRD	SHLD shifts "dest" to the left "count" times and the bit positions opened are filled with the most significant bits of "src". SHRD shifts "dest" to the right "count" times and the bit positions opened are filled with the least significant bits of the second operand. Only the 5 lower bits of "count" are used. Modified: AF, CF, OF, PF, SF, ZF
STC	Sets the Carry Flag to 1.
STD	Sets the Direction Flag to 1 causing string instructions to auto-decrement SI and DI instead of auto-increment.
SUB	The source is subtracted from the destination and the result is stored in the destination. Modified: AF, CF, OF, PF, SF, ZF
XADD	Exchanges the first operand with the second operand, then loads the sum of the two values into the destination operand.
XOR	Performs a bitwise exclusive OR of the operands and returns the result in the destination. Modified: AF, CF, OF, PF, SF, ZF

Data Transfer

Mnemonic	Description
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
LAHF	Copies flags AF, CF, PF, SF, and ZF into AH
LAR	The high byte of the of the destination register is overwritten by the value of the access rights byte and the low order byte is zeroed depending on the selection in the source operand. The Zero Flag is set if the load operation is successful.
LDS	Loads 32-bit pointer from memory source to destination register and DS. The offset is placed in the destination register and the segment is placed in DS. To use this instruction the word at the lower memory address must contain the offset and the word at the higher address must contain the segment. This simplifies the loading of far pointers from the stack and the interrupt vector table.
LEA	Transfers offset address of "src" to the destination register.
LES	Loads 32-bit pointer from memory source to destination register and ES. The offset is placed in the destination register and the segment is placed in ES. To use this instruction the word at the lower memory address must contain the offset and the word at the higher address must contain the segment. This simplifies the loading of far pointers from the stack and the interrupt vector table.
LFENCE	Load fence; Serializes load operations
LFS	Loads 32-bit pointer from memory source to destination register and FS. The offset is placed in the destination register and the segment is placed in FS. To use this instruction the word at the lower memory address must contain the offset and the word at the higher address must contain the segment. This simplifies the loading of far pointers from the stack and the interrupt vector table.
LGDT	Loads a value from an operand into the Global Descriptor Table (GDT) register.
LIDT	Loads a value from an operand into the Interrupt Descriptor Table (IDT) register.
LGS	Loads 32-bit pointer from memory source to destination register and GS. The offset is placed in the destination register and the segment is placed in GS. To use this instruction the word at the lower memory address must contain the offset and the word at the higher address must contain the segment. This simplifies the loading of far pointers from the stack and the interrupt vector table.
LLDT	Loads a value from an operand into the Local Descriptor Table Register (LDTR).
LMSW	Loads the Machine Status Word (MSW) from data found at "src".

Created by Devyn Collier Johnson <DevynCJohnson@Gmail.com> (2016v2) More cheatsheets at DCJTech.info

x86 Assembly

LODS	Transfers string element addressed by DS:SI (even if an operand is supplied) to the accumulator. SI is incremented based on the size of the operand or based on the instruction used. If the Direction Flag is set SI is decremented, if the Direction Flag is clear SI is incremented. Use with REP prefixes.
LODSQ	LOaD String Quadword
LSL	Loads the segment limit of a selector into the destination register if the selector is valid and visible at the current privilege level. If loading is successful the Zero Flag is set, otherwise it is cleared.
LSS	Loads 32-bit pointer from memory source to destination register and SS. The offset is placed in the destination register and the segment is placed in SS. To use this instruction the word at the lower memory address must contain the offset and the word at the higher address must contain the segment. This simplifies the loading of far pointers from the stack and the interrupt vector table.
LTR	Loads the current task register with the value specified in "src".
MASKMOVDQU	Masked Move of Double Quadword Unaligned; Stores selected bytes from the source operand (first operand) into a 128-bit memory location
MASKMOVQ	Masked Move of Quadword; Selectively write bytes from MM1 to memory location using the byte mask in MM2
MFENCE	Memory Fence; Performs a serializing operation on all load and store instructions that were issued prior the MFENCE instruction.
MOV	Copies byte or word from the source operand to the destination operand.
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.
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
MOVS	Copies data from addressed by DS:SI (even if operands are given) to the location ES:DI destination and updates SI and DI based on the size of the operand or instruction used. SI and DI are incremented when the Direction Flag is cleared and decremented when the Direction Flag is Set.
MOVSX	Copies the value of the source operand to the destination register with the sign extended.
MOVSXD	MOV with Sign Extend 32-bit to 64-bit
MOVZX	Copies the value of the source operand to the destination register with the zeros extended.
OUT	Transfers byte in AL,word in AX or dword in EAX to the specified hardware port address. If the port number is in the range of 0-255 it can be specified as an immediate. If greater than 255 then the port number must be specified in DX. Since the PC only decodes 10 bits of the port address, values over 1023 can only be decoded by third party vendor equipment and also map to the port range 0-1023.
OUTS	Transfers a byte, word or doubleword from "src" to the hardware port specified in DX. For instructions with no operands the "src" is located at DS:SI and SI is incremented or decremented by the size of the operand or the size dictated by the instruction format. When the Direction Flag is set SI is decremented, when clear, SI is incremented. If the port number is in the range of 0-255 it can be specified as an immediate. If greater than 255 then the port number must be specified in DX. Since the PC only decodes 10 bits of the port address, values over 1023 can only be decoded by third party vendor equipment and also map to the port range 0-1023.
POP	Transfers word at the current stack top (SS:SP) to the destination then increments SP by two to point to the new stack top. CS is not a valid destination.
POPA/POPAD	Pops the top 8 words off the stack into the 8 general purpose 16/32 bit registers. Registers are popped in the following order: (E)DI, (E)SI, (E)BP, (E)SP, (E)DX, (E)CX and (E)AX. The (E)SP value popped from the stack is actually discarded.
POPF/POPFD	Pops word/doubleword from stack into the Flags Register and then increments SP by 2 (for POPF) or 4 (for POPFD).
POPFQ	POP RFLAGS Register
PREFETCH0	Prefetch into all cache levels
PREFETCH1	Prefetch into all cache levels EXCEPT L1
PREFETCH2	Prefetch into all cache levels EXCEPT L1 and L2
PREFETCHNTA	Prefetch into all cache levels to non-temporal cache structure
PUSH	Decrements SP by the size of the operand (two or four, byte values are sign extended) and transfers one word from source to the stack top (SS:SP).
PUSHA/PUSHAD	Pushes all general purpose registers onto the stack in the following order: (E)AX, (E)CX, (E)DX, (E)BX, (E)SP, (E)BP, (E)SI, (E)DI. The value of SP is the value before the actual push of SP.
PUSHF/PUSHFD	Transfers the Flags Register onto the stack. PUSHF saves a 16 bit value while PUSHFD saves a 32 bit value.
PUSHFQ	PUSH RFLAGS Register
RDMSR	Load MSR specified by ECX into EDX:EAX.
RDPMC	Read the PMC [Performance Monitoring Counter]; Specified in the ECX register into registers EDX:EAX
RDTSC	Returns the number of processor ticks since the processor being "ONLINE" (since the last power on of system).
RDTSCP	ReaD Time Stamp Counter and Processor ID
SAHF	Transfers bits 0-7 of AH into the Flags Registers AF, CF, PF, SF, and ZF.
SFENCE	Processor hint to make sure all store operations that took place prior to the SFENCE call are globally visible
SGDT	Stores the Global Descriptor Table (GDT) Register into the specified operand.
SIDT	Stores the Interrupt Descriptor Table (IDT) Register into the specified operand.
SLDT	Stores the Local Descriptor Table (LDT) Register into the specified operand.

Created by Devyn Collier Johnson <DevynCJohnson@Gmail.com> (2016v2) More cheatsheets at DCJTech.info

x86 Assembly

SMSW	Store Machine Status Word (MSW) into "dest".
STOS	Stores value in accumulator to location at ES:(E)DI (even if operand is given). (E)DI is incremented/decremented based on the size of the operand (or instruction format) and the state of the Direction Flag.
STOSQ	STOre String Quadword
STR	Stores the current Task Register to the specified operand.
SWAPGS	Exchange GS base with KernelGSBase MSR
WRMSR	Write the value in EDX:EAX to MSR specified by ECX.
XCHG	Exchanges contents of source and destination.
XLAT/XLATB	Replaces the byte in AL with byte from a user table addressed by BX. The original value of AL is the index into the translate table. The best way to describe this is MOV AL,[BX+AL]

x87 Floating-Point Instructions

Introduced in 8087, 80287, and Pentium

Mnemonic	Description	Mnemonic	Description
F2XM1	2^x-1	**FINIT**	Initialize floating point processor
FABS	Absolute value	**FIST**	Store integer
FADD	Add	**FISTP**	Store integer and pop
FADDP	Add and pop	**FISUB**	Integer subtract
FBLD	Load BCD	**FISUBR**	Integer subtract reversed
FBSTP	Store BCD and pop	**FLD**	Floating point load
FCHS	Change sign	**FLD1**	Load 1.0 onto stack
FCLEX	Clear exceptions	**FLDCW**	Load control word
FCOM	Compare	**FLDENV**	Load environment state
FCOMP	Compare and pop	**FLDENVD**	Load environment state, 32-bit
FCOMPP	Compare and pop twice	**FLDENVW**	Load environment state, 16-bit
FCOS	Cosine	**FLDL2E**	Load $\log_2(e)$ onto stack
FDECSTP	Decrement floating-point stack pointer	**FLDL2T**	Load $\log_2(10)$ onto stack
FDISI	Divide	**FLDLG2**	Load $\log_{10}(2)$ onto stack
FDIV	Divide	**FLDLN2**	Load ln(2) onto stack
FDIVP	Divide and pop	**FLDPI**	Load π onto stack
FDIVR	Divide reversed	**FLDZ**	Load 0.0 onto stack
FDIVRP	Divide reversed and pop	**FMUL**	Multiply
FENI	Enable interrupts; 8087 only, otherwise FNOP	**FMULP**	Multiply and pop
FFREE	Free Register	**FNCLEX**	Clear exceptions, no wait
FIADD	Integer add	**FNDISI**	Disable interrupts, no wait; 8087 only, otherwise FNOP
FICOM	Integer compare	**FNENI**	Enable interrupts, no wait; 8087 only, otherwise FNOP
FICOMP	Integer compare and pop	**FNINIT**	Initialize floating point processor, no wait
FIDIV	Integer divide	**FSIN**	Sine
FIDIVR	Integer divide reversed	**FSINCOS**	Sine and cosine
FILD	Load integer	**FSQRT**	Square root
FIMUL	Integer multiply	**FST**	Floating point store
FINCSTP	Increment floating point stack pointer	**FSTCW**	Store control word
FINIT	Initialize floating point processor	**FSTENV**	Store FPU environment
FIST	Store integer	**FSTENVD**	Store FPU environment, 32-bit
FISTP	Store integer and pop	**FSTENVD**	Store FPU environment, 32-bit
FISUB	Integer subtract	**FSTENVW**	Store FPU environment, 16-bit
FISUBR	Integer subtract reversed	**FSTP**	Store and pop
FLD	Floating point load	**FSTSW**	Store status word
FLD1	Load 1.0 onto stack	**FSUB**	Subtract

Created by Devyn Collier Johnson <DevynCJohnson@Gmail.com> (2016v2) More cheatsheets at DCJTech.info

x86 Assembly

Mnemonic	Description	Mnemonic	Description
FLDCW	Load control word	**FSUBP**	Subtract and pop
FLDENV	Load environment state	**FSUBR**	Reverse subtract
FLDENVD	Load environment state, 32-bit	**FSUBRP**	Reverse subtract and pop
FLDENVW	Load environment state, 16-bit	**FTST**	Test for zero
FLDL2E	Load $\log_2(e)$ onto stack	**FUCOM**	Unordered compare
FLDL2T	Load $\log_2(10)$ onto stack	**FUCOMP**	Unordered compare and pop
FLDLG2	Load $\log_{10}(2)$ onto stack	**FUCOMPP**	Unordered compare and pop twice
FLDLN2	Load ln(2) onto stack	**FWAIT**	Wait while FPU is executing
FLDPI	Load π onto stack	**FXAM**	Examine condition flags
FLDZ	Load 0.0 onto stack	**FXCH**	Exchange registers
FMUL	Multiply	**FXTRACT**	Extract exponent and significand
FMULP	Multiply and pop	**FYL2X**	$y*\log_2(x)$
FNCLEX	Clear exceptions, no wait	**FYL2XP1**	$y*\log_2(x+1)$
FNDISI	Disable interrupts, no wait; 8087 only, otherwise FNOP	**FRNDINT**	Round to integer
FNENI	Enable interrupts, no wait; 8087 only, otherwise FNOP	**FRSTOR**	Restore saved state
FNINIT	Initialize floating point processor, no wait	**FRSTORD**	Restore saved state, 32-bit
FNOP	No operation	**FRSTORW**	Restore saved state
FNSAVE	Save FPU state, no wait, 8-bit	**FSAVE**	Save FPU state
FNSAVEW	Save FPU state, no wait, 16-bit	**FSAVED**	Save FPU state, 32-bit
FNSTCW	Store control word, no wait	**FSAVEW**	Save FPU state, 16-bit
FNSTENV	Store FPU environment, no wait	**FSCALE**	Scale by factor of 2
FNSTENVW	Store FPU environment, no wait, 16-bit	**FSETPM**	Set protected mode; 80287 only, otherwise FNOP
FNSTSW	Store status word, no wait	**FPREM1**	Partial remainder
FPATAN	Partial arctangent	**FPTAN**	Partial tangent
FPREM	Partial remainder		

Variants Introduced in the Pentium-Pro

FCMOV: FCMOVB, FCMOVBE, FCMOVE, FCMOVNB, FCMOVNBE, FCMOVNE, FCMOVNU, FCMOVU

FCOMI: FCOMI, FCOMIP (Pop FP stack), FUCOMI, FUCOMIP (Pop FP stack); Compare and move results to Integer flags

MMX Instructions

Introduced in the Pentium-P5; Equivalent to AMD's *3DNow!* and ARM's *iwMMXt*

Mnemonic	Description	Mnemonic	Description
EMMS	Empty MMX Technology State; Marks all x87 FPU registers for use by FPU	**PMADDWD**	Multiply packed word integers, add adjacent doubleword results
MOVD	Move doubleword	**PMULHW**	Multiply packed signed word integers, store high 16 bit results
MOVQ	Move quadword	**PMULLW**	Multiply packed signed word integers, store low 16 bit results
PACKSSDW	Pack doubleword to word (signed with saturation)	**PSLLW**	Shift left word, shift in zeros
PACKSSWB	Pack word to byte (signed with saturation)	**PSLLD**	Shift left doubleword, shift in zeros
PACKUSWB	Pack word to byte (signed with unsaturation)	**PSLLQ**	Shift left quadword, shift in zeros
PADDB	Add packed byte integers	**PSRAD**	Shift right doubleword, shift in sign bits
PADDW	Add packed word integers	**PSRAW**	Shift right word, shift in sign bits
PADDD	Add packed doubleword integers	**PSRLW**	Shift right word, shift in zeros
PADDSB	Add packed signed byte integers and saturate	**PSRLD**	Shift right doubleword, shift in zeros
PADDSW	Add packed signed word integers and saturate	**PSRLQ**	Shift right quadword, shift in zeros
PADDUSB	Add packed unsigned byte integers and saturate	**PSUBB**	Subtract packed byte integers
PADDUSW	Add packed unsigned word integers and saturate	**PSUBW**	Subtract packed word integers
PAND	Bitwise AND	**PSUBD**	Subtract packed doubleword integers
PANDN	Bitwise AND NOT	**PSUBSB**	Subtract packed signed byte integers with saturation

Created by Devyn Collier Johnson <DevynCJohnson@Gmail.com> (2016v2) More cheatsheets at DCJTech.info

x86 Assembly

Mnemonic	Description	Mnemonic	Description
POR	Bitwise OR	**PSUBSW**	Subtract packed signed word integers with saturation
PXOR	Bitwise XOR	**PSUBUSB**	Subtract packed unsigned byte integers with saturation
PCMPEQB	Compare packed byte integers for equality	**PSUBUSW**	Subtract packed unsigned word integers with saturation
PCMPEQW	Compare packed word integers for equality	**PUNPCKHBW**	Unpack and interleave high-order bytes
PCMPEQD	Compare packed doubleword integers for equality	**PUNPCKHWD**	Unpack and interleave high-order words
PCMPGTB	Compare packed signed byte integers for greater than	**PUNPCKHDQ**	Unpack and interleave high-order doublewords
PCMPGTW	Compare packed signed word integers for greater than	**PUNPCKLBW**	Unpack and interleave low-order bytes
PCMPGTD	Compare packed signed doubleword integers for greater than	**PUNPCKLDQ**	Unpack and interleave low-order words
		PUNPCKLWD	Unpack and interleave low-order doublewords

SSE Instructions

Introduced in the Pentium-III

- **SSE 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 Integer Instructions:** ANDNPS, ANDPS, ORPS, PAVGB, PAVGW, PEXTRW, PINSRW, PMAXSW, PMAXUB, PMINSW, PMINUB, PMOVMSKB, PMULHUW, PSADBW, PSHUFW, XORPS

SSE2 (Willamette) Instructions

Introduced in the Pentium-4

- **SSE2 Floating-Point Instructions:** 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

SSE3 (Prescott) Instructions

Introduced in the Pentium-4 (Prescott revision)

- **SSE3 Floating-Point Instructions:** ADDSUBPD, ADDSUBPS, HADDPD, HADDPS, HSUBPD, HSUBPS, MOVDDUP, MOVSHDUP, MOVSLDUP
- **SSE3 Integer Instructions:** LDDQU
- **SSE3 Float-2-Int Conversions:** FISTTP; x87 to integer truncation conversion regardless of status word
- **SSE3 (Intel Specific):** MONITOR, MWAIT; (Optimize multi-threaded applications, giving Hyper-Threading better performance)

SSSE3 (Merom) Instructions

Introduced in Woodcrest and Bobcat

Mnemonic	Description
PSIGNB, PSIGNW, PSIGND	Packed Sign; Negate the elements of a register of bytes, words or dwords if the sign of the corresponding elements of another register is negative.
PABSB, PABSW, PABSD	Packed Absolute Value; Fill the elements of a register of bytes, words or dwords with the absolute values of the elements of another register
PALIGNR	Packed Align Right; Take two registers, concatenate their values, and pull out a register-length section from an offset given by an immediate value encoded in the instruction.
PSHUFB	Packed Shuffle Bytes
PMULHRSW	Packed Multiply High with Round and Scale; Treat the sixteen-bit words in registers A and B as signed 15-bit fixed-point numbers between −1 and 1 (e.g. 0x4000 is treated as 0.5 and 0xa000 as −0.75), and multiply them together with correct rounding.
PMADDUBSW	Multiply and Add Packed Signed and Unsigned Bytes
PHSUBW, PHSUBD	Packed Horizontal Subtract (Words or Doublewords); Takes registers A = [a0 a1 a2 …] and B = [b0 b1 b2 …] and outputs [a0−a1 a2−a3 … b0−b1 b2−b3 …]
PHSUBSW	Packed Horizontal Subtract and Saturate Words
PHADDW, PHADDD	Packed Horizontal Add (Words or Doublewords); Takes registers A = [a0 a1 a2 …] and B = [b0 b1 b2 …] and outputs [a0+a1 a2+a3 … b0+b1 b2+b3 …]
PHADDSW	Packed Horizontal Add and Saturate Words

Each instruction can act on 64-bit MMX or 128-bit XMM registers

Created by Devyn Collier Johnson <DevynCJohnson@Gmail.com> (2016v2) More cheatsheets at DCJTech.info

x86 Assembly

SSE4.1 (Penryn) Instructions

Introduced in the Intel Core microarchitecture, Bulldozer, AMD K10 (K8L), and VIA Nano-based processors

Mnemonic	Description
MPSADBW	Compute eight offset sums of absolute differences, four at a time
PHMINPOSUW	Sets the bottom unsigned 16-bit word of the destination to the smallest unsigned 16-bit word in the source, and the next-from-bottom to the index of that word in the source.
PMULDQ	Packed signed multiplication on two sets of two out of four packed integers, the 1st and 3rd per packed 4, giving two packed 64-bit results.
PMULLD	Packed signed multiplication, four packed sets of 32-bit integers multiplied to give 4 packed 32-bit results.
DPPS, DPPD	Dot product for AOS (Array of Structs) data. This takes an immediate operand consisting of four (or two for DPPD) bits to select which of the entries in the input to multiply and accumulate, and another four (or two for DPPD) to select whether to put 0 or the dot-product in the appropriate field of the output.
BLENDPS, BLENDPD, BLENDVPS, BLENDVPD, PBLENDVB, PBLENDW	Conditional copying of elements in one location with another, based (for non-V form) on the bits in an immediate operand, and (for V form) on the bits in register XMM0.
PMINSB, PMAXSB, PMINUW, PMAXUW, PMINUD, PMAXUD, PMINSD, PMAXSD	Packed minimum/maximum for different integer operand types
ROUNDPS, ROUNDSS, ROUNDPD, ROUNDSD	Round values in a floating-point register to integers, using one of four rounding modes specified by an immediate operand
INSERTPS, PINSRB, PINSRD, PINSRQ, EXTRACTPS, PEXTRB, PEXTRD, PEXTRQ	The INSERTPS and PINSR instructions read 8, 16 or 32 bits from an x86 register or memory location and inserts it into a field in the destination register given by an immediate operand. EXTRACTPS and PEXTR read a field from the source register and insert it into an x86 register or memory location.
PMOVSXBW, PMOVZXBW, PMOVSXBD, PMOVZXBD, PMOVSXBQ, PMOVZXBQ, PMOVSXWD, PMOVZXWD, PMOVSXWQ, PMOVZXWQ, PMOVSXDQ,PMOVZXDQ	Packed sign/zero extension to wider types
PTEST	This is similar to the TEST instruction, in that it sets the Z flag to the result of an AND between its operands: ZF is set, if DEST AND SRC is equal to 0. Additionally it sets the C flag if (NOT DEST) AND SRC equals zero.
PCMPEQQ	Quadword (64 bits) compare for equality
PACKUSDW	Convert signed DWORDs into unsigned WORDs with saturation
MOVNTDQA	Efficient read from write-combining memory area into SSE register

- **SSE4.1 Floating-Point Instructions:** DPPS, DPPD, BLENDPS, BLENDPD, BLENDVPS, BLENDVPD, ROUNDPS, ROUNDSS, ROUNDPD, ROUNDSD, INSERTPS, EXTRACTPS
- **SSE4.1 Integer Instructions:** MPSADBW, PHMINPOSUW, PMULLD, PMULDQ, PBLENDVB, PBLENDW, PMINSB, PMAXSB, PMINUW, PMAXUW, PMINUD, PMAXUD, PMINSD, PMAXSD, PINSRB, PINSRD/PINSRQ, PEXTRB, PEXTRW, PEXTRD, PEXTRQ, PMOVSXBW, PMOVZXBW, PMOVSXBD, PMOVZXBD, PMOVSXBQ, PMOVZXBQ, PMOVSXWD, PMOVZXWD, PMOVSXWQ, PMOVZXWQ, PMOVSXDQ, PMOVZXDQ, PTEST, PCMPEQQ, PACKUSDW, MOVNTDQA

SSE4.2 Instructions

Introduced in the Nehalem-based Core i7 and Bulldozer

Mnemonic	Description
CRC32	Accumulate CRC32C value using the polynomial 0x11EDC6F41 (or, without the high order bit, 0x1EDC6F41)
PCMPESTRI	Packed Compare Explicit Length Strings, Return Index
PCMPESTRM	Packed Compare Explicit Length Strings, Return Mask
PCMPISTRI	Packed Compare Implicit Length Strings, Return Index
PCMPISTRM	Packed Compare Implicit Length Strings, Return Mask
PCMPGTQ	Compare Packed Signed 64-bit data For Greater Than

SSE4a Instructions

Introduced in AMD's Barcelona microarchitecture

Mnemonic	Description
EXTRQ	Extracts a particular set of bits from a register and moves them to the register's least significant position
INSERTQ	Inserts field from a source register to a destination register
MOVNTSD	Write the least significant 32 bits of a register to memory using the non-temporal hint. For example, a loop that performs scalar single-precision floating point math on a large array can use the SSE registers and MOVNTSS to store results to memory.
MOVNTSS	Write the lower 64 bits of a register to memory using the non-temporal hint. This instruction can be used for similar purposes as the MOVNTSS instruction, but typically for double-precision floating point data.

Created by Devyn Collier Johnson <DevynCJohnson@Gmail.com> (2016v2) More cheatsheets at DCJTech.info

x86 Assembly

Advanced Bit Manipulation (ABM) Instructions

Introduced in AMD's Barcelona microarchitecture

Mnemonic	Description
LZCNT	Leading zero count; Introduced in Haswell
POPCNT	Population count (count number of bits set to 1); Introduced in Nehalem

Bit Manipulation Instruction Set 1 (BMI1) Instructions

Introduced in Haswell, Jaguar, and Piledriver

Mnemonic	Description
ANDN	Logical and not; (~x & y)
BEXTR	Bit field extract (with register); ((src >> start) & ((1 << len)-1))
BLSI	Extract lowest set isolated bit; (x & -x)
BLSMSK	Get mask up to lowest set bit; (x ^ (x - 1))
BLSR	Reset lowest set bit; (x & (x - 1))
TZCNT	Count the number of trailing zero bits

Bit Manipulation Instruction Set 2 (BMI2) Instructions

Introduced in Haswell and Excavator

Mnemonic	Description
BZHI	Zero high bits starting with specified bit position
MULX	Unsigned multiply without affecting flags, and arbitrary destination registers
PDEP	Parallel bits deposit
PEXT	Parallel bits extract
RORX	Rotate right logical without affecting flags
SARX	Shift arithmetic right without affecting flags
SHRX	Shift logical right without affecting flags
SHLX	Shift logical left without affecting flags

Trailing Bit Manipulation (TBM) Instructions

Mnemonic	Description
BEXTR	Bit field extract (with immediate); ((src >> start) & ((1 << len)-1))
BLCFILL	Fill from lowest clear bit; (x & (x + 1))
BLCI	Isolate lowest clear bit; (x | ~(x + 1))
BLCIC	Isolate lowest clear bit and complement; (~x & (x + 1))
BLCMASK	Mask from lowest clear bit; (x ^ (x + 1))
BLCS	Set lowest clear bit; (x | (x + 1))
BLSFILL	Fill from lowest set bit; (x | (x - 1))
BLSIC	Isolate lowest set bit and complement; (~x | (x - 1))
T1MSKC	Inverse mask from trailing ones; (~x | (x + 1))
TZMSK	Mask from trailing zeros; (~x & (x - 1))

Multi-Precision Add-Carry Instruction Extensions (ADX)

Introduced in Broadwell

Mnemonic	Description
ADCX	Adds two unsigned integers plus carry, reading the carry from the carry flag and if necessary setting it there. Does not affect other flags than the carry.
ADOX	Adds two unsigned integers plus carry, reading the carry from the overflow flag and if necessary setting it there. Does not affect other flags than the overflow.

Created by Devyn Collier Johnson <DevynCJohnson@Gmail.com> (2016v2) More cheatsheets at DCJTech.info

x86 Assembly

Carry-less Multiplication (CLMUL) Instructions

Introduced in Westmere and Bulldozer

Mnemonic (Intel Syntax)	Description
PCLMULQDQ xmmreg,xmmrm,imm	Perform a carry-less multiplication of two 64-bit polynomials over the finite field GF(2)
PCLMULLQLQDQ xmmreg,xmmrm	Multiply the low halves of the two registers
PCLMULHQLQDQ xmmreg,xmmrm	Multiply the high half of the destination register by the low half of the source register
PCLMULLQHQDQ xmmreg,xmmrm	Multiply the low half of the destination register by the high half of the source register
PCLMULHQHQDQ xmmreg,xmmrm	Multiply the high halves of the two registers

NOTE: Finite field (GF(2**k)) multiplication can be implemented more efficiently

FMA/FMA3 Instructions

Introduced in Piledriver, Haswell, and Broadwell

Mnemonic	Description
VFMADDPD	Fused Multiply-Add of Packed Double-Precision Floating-Point Values; VFMADDPD xmm0, xmm1, xmm2, xmm3
VFMADDPS	Fused Multiply-Add of Packed Single-Precision Floating-Point Values
VFMADDSD	Fused Multiply-Add of Scalar Double-Precision Floating-Point Values
VFMADDSS	Fused Multiply-Add of Scalar Single-Precision Floating-Point Values
VFMADDSUBPD	Fused Multiply-Alternating Add/Subtract of Packed Double-Precision Floating-Point Values
VFMADDSUBPS	Fused Multiply-Alternating Add/Subtract of Packed Single-Precision Floating-Point Values
VFMSUBADDPD	Fused Multiply-Alternating Subtract/Add of Packed Double-Precision Floating-Point Values
VFMSUBADDPS	Fused Multiply-Alternating Subtract/Add of Packed Single-Precision Floating-Point Values
VFMSUBPD	Fused Multiply-Subtract of Packed Double-Precision Floating-Point Values
VFMSUBPS	Fused Multiply-Subtract of Packed Single-Precision Floating-Point Values
VFMSUBSD	Fused Multiply-Subtract of Scalar Double-Precision Floating-Point Values
VFMSUBSS	Fused Multiply-Subtract of Scalar Single-Precision Floating-Point Values
VFNMADDPD	Fused Negative Multiply-Add of Packed Double-Precision Floating-Point Values
VFNMADDPS	Fused Negative Multiply-Add of Packed Single-Precision Floating-Point Values
VFNMADDSD	Fused Negative Multiply-Add of Scalar Double-Precision Floating-Point Values
VFNMADDSS	Fused Negative Multiply-Add of Scalar Single-Precision Floating-Point Values
VFNMSUBPD	Fused Negative Multiply-Subtract of Packed Double-Precision Floating-Point Values
VFNMSUBPS	Fused Negative Multiply-Subtract of Packed Single-Precision Floating-Point Values
VFNMSUBSD	Fused Negative Multiply-Subtract of Scalar Double-Precision Floating-Point Values
VFNMSUBSS	Fused Negative Multiply-Subtract of Scalar Single-Precision Floating-Point Values

NOTE: VFMADDPDx uses XMM; VFMADDPDy uses YMM

FMA4 Instructions

Introduced in Bulldozer

Mnemonic	Operands
VFMADDPDx	xmm, xmm, xmm/m128, xmm/m128
VFMADDPDy	ymm, ymm, ymm/m256, ymm/m256
VFMADDPSx	xmm, xmm, xmm/m128, xmm/m128
VFMADDPSy	ymm, ymm, ymm/m256, ymm/m256
VFMADDSD	xmm, xmm, xmm/m64, xmm/m64
VFMADDSS	xmm, xmm, xmm/m32, xmm/m32

NOTE: FMA4 uses four operands while FMA3 uses three operands

Created by Devyn Collier Johnson <DevynCJohnson@Gmail.com> (2016v2) More cheatsheets at DCJTech.info

x86 Assembly

AVX Instructions

Introduced in Sandy Bridge and Bulldozer

Mnemonic	Description
VBROADCASTSS, VBROADCASTSD, VBROADCASTF128	Copy a 32, 64, or 128 bit memory operand to all elements of a XMM or YMM vector register
VINSERTF128	Replaces either the lower half or the upper half of a 256-bit YMM register with the value of a 128-bit source operand. The other half of the destination is unchanged.
VEXTRACTF128	Extracts either the lower half or the upper half of a 256-bit YMM register and copies the value to a 128-bit destination operand
VMASKMOVPS, VMASKMOVPD	Conditionally reads any number of elements from a SIMD vector memory operand into a destination register, leaving the remaining vector elements unread and setting the corresponding elements in the destination register to zero
VPERMILPS, VPERMILPD	Permute In-Lane. Shuffle the 32-bit or 64-bit vector elements of one input operand.
VPERM2F128	Shuffle the four 128-bit vector elements of two 256-bit source operands into a 256-bit destination operand, with an immediate constant as selector
VZEROALL	Set all YMM registers to zero and tag them as unused; Used when switching between 128-bit and 256-bit
VZEROUPPER	Set the upper half of all YMM registers to zero; Used when switching between 128-bit and 256-bit

AVX2 Instructions

Introduced in Haswell and Carrizo

Mnemonic	Description
VBROADCASTSS, VBROADCASTSD	Copy a 32-bit or 64-bit register operand to all elements of a XMM or YMM vector register. These are register versions of the same instructions in AVX1. There is no 128-bit version however, but the same effect can be simply achieved using VINSERTF128.
VPBROADCASTB, VPBROADCASTW, VPBROADCASTD, VPBROADCASTQ	Copy an 8, 16, 32 or 64-bit integer register or memory operand to all elements of a XMM or YMM vector register
VBROADCASTI128	Copy a 128-bit memory operand to all elements of a YMM register
VINSERTI128	Replaces either the lower half or the upper half of a 256-bit YMM register with the value of a 128-bit source operand. The other half of the destination is unchanged
VEXTRACTI128	Extracts either the lower half or the upper half of a 256-bit YMM register and copies the value to a 128-bit destination operand
VGATHERDPD, VGATHERQPD, VGATHERDPS, VGATHERQPS	Gathers single or double precision floating point values using either 32 or 64-bit indices and scale
VPGATHERDD, VPGATHERDQ, VPGATHERQD, VPGATHERQQ	Gathers 32 or 64-bit integer values using either 32 or 64-bit indices and scale
VPMASKMOVD, VPMASKMOVQ	Conditionally reads any number of elements from a SIMD vector memory operand into a destination register, leaving the remaining vector elements unread and setting the corresponding elements in the destination register to zero
VPERMPS, VPERMD	Shuffle the eight 32-bit vector elements of one 256-bit source operand into a 256-bit destination operand, with a register or memory operand as selector
VPERMPD, VPERMQ	Shuffle the four 64-bit vector elements of one 256-bit source operand into a 256-bit destination operand, with a register or memory operand as selector
VPERM2I128	Shuffle the four 128-bit vector elements of two 256-bit source operands into a 256-bit destination operand, with an immediate constant as selector
VPBLENDD	Doubleword immediate version of the PBLEND instructions from SSE4
VPSLLVD, VPSLLVQ	Shift left logical. Allows variable shifts where each element is shifted according to the packed input.
VPSRLVD, VPSRLVQ	Shift right logical. Allows variable shifts where each element is shifted according to the packed input.
VPSRAVD	Shift right arithmetically. Allows variable shifts where each element is shifted according to the packed input.

Created by Devyn Collier Johnson <DevynCJohnson@Gmail.com> (2016v2) More cheatsheets at DCJTech.info

x86 Assembly

AES Instructions

Introduced in Westmere and Bulldozer

Mnemonic	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
PCLMULQDQ	Carryless multiply

SHA Instructions

Introduced in Intel's Goldmont microarchitecture

Mnemonic	Description
SHA1RNDS4	SHA-1
SHA1NEXTE	SHA-1
SHA1MSG1	SHA-1
SHA1MSG2	SHA-1
SHA256RNDS2	SHA-256
SHA256MSG1	SHA-256
SHA256MSG2	SHA-256

XOP Instructions

Introduced in Bulldozer

Mnemonic	Description	Mnemonic	Description
Integer Vector Multiply-Accumulate		**Vector Conditional Move**	
VPMACSWW, VPMACSSWW	Multiply Accumulate (with Saturation) Word to Word	**VPCMOV**	Vector Conditional Move
VPMACSWD, VPMACSSWD	Multiply Accumulate (with Saturation) Low Word to Dword	**Integer Vector Shift and Rotate**	
VPMACSDD, VPMACSSDD	Multiply Accumulate (with Saturation) Dword to Dword	**VPROTB**	Packed Rotate Bytes
VPMACSDQL, VPMACSSDQL	Multiply Accumulate (with Saturation) Low Dword to Qword	**VPROTW**	Packed Rotate Words
VPMACSDQH, VPMACSSDQH	Multiply Accumulate (with Saturation) High Dword to Qword	**VPROTD**	Packed Rotate Doublewords
VPMADCSWD, VPMADCSSWD	Multiply Add Accumulate (with Saturation) Word to Dword	**VPROTQ**	Packed Rotate Quadwords
Integer Vector Horizontal Addition		**VPSHAB**	Packed Shift Arithmetic Bytes
VPHADDBW, VPHADDUBW	Horizontal add two bytes to word	**VPSHAW**	Packed Shift Arithmetic Words
VPHADDBD, VPHADDUBD	Horizontal add four bytes to dword	**VPSHAD**	Packed Shift Arithmetic Doublewords
VPHADDBQ, VPHADDUBQ	Horizontal add eight bytes to quadword	**VPSHAQ**	Packed Shift Arithmetic Quadwords
VPHADDWD, VPHADDUWD	Horizontal add two signed/unsigned words to dword	**VPSHLB**	Packed Shift Logical Bytes
VPHADDWQ, VPHADDUWQ	Horizontal add four words to quadword	**VPSHLW**	Packed Shift Logical Words
VPHADDDQ, VPHADDUDQ	Horizontal add two dwords to quadword	**VPSHLD**	Packed Shift Logical Doublewords
VPHSUBBW	Horizontal subtract two signed bytes to word	**VPSHLQ**	Packed Shift Logical Quadwords
VPHSUBWD	Horizontal subtract two signed words to dword	**Vector Permute**	
VPHSUBDQ	Horizontal subtract two signed dwords to qword	**VPPERM**	Packed Permute Byte
Integer Vector Compare		**VPPERMIL2PD**	Permute Two-Source Double-Precision Floating-Point
VPCOMB	Compare Vector Signed Bytes	**VPPERMIL2PS**	Permute Two-Source Single-Precision Floating-Point
VPCOMW	Compare Vector Signed Words	**Floating-point Fraction Extraction**	
VPCOMD	Compare Vector Signed Doublewords	**VFRCZPD**	Extract Fraction Packed Double-Precision Floating-Point
VPCOMQ	Compare Vector Signed Quadwords	**VFRCZPS**	Extract Fraction Packed Single-Precision Floating-Point
VPCOMUB	Compare Vector Unsigned Bytes	**VFRCZSD**	Extract Fraction Scalar Double-Precision Float
VPCOMUW	Compare Vector Unsigned Words	**VFRCZSS**	Extract Fraction Scalar Float
VPCOMUD	Compare Vector Unsigned Doublewords		
VPCOMUQ	Compare Vector Unsigned Quadwords		

Created by Devyn Collier Johnson <DevynCJohnson@Gmail.com> (2016v2) More cheatsheets at DCJTech.info

x86 Assembly

XOP Comparison Immediates

Introduced in Bulldozer

Immediate	Description
000	Less Than
001	Less Than or Equal
010	Greater Than
011	Greater Than or Equal
100	Equal
101	Not Equal
110	False
111	True

Miscellaneous Instructions

Intel VT-x: VMPTRLD, VMPTRST, VMCLEAR, VMREAD, VMWRITE, VMCALL, VMLAUNCH, VMRESUME, VMXOFF, VMXON
RDRAND (Ivy Bridge)
RDSEED (Introduced in Broadwell and Zen)
UMOV (80386/80486 ICE processors only)
LOADALL, LOADALLD, UD1, SALC
SYSCALL (AMD K6; equivalent to SYSENTER)
SYSRET (AMD K6; equivalent to SYSEXIT)
FFREEP performs FFREE ST(i) and pop stack

F16C/CVT16 Instructions

Introduced in Bulldozer

Mnemonic (Intel Syntax)	Description
VCVTPH2PS xmmreg,xmmrm64	Convert four half-precision floating point values in memory or the bottom half of an XMM register to four single-precision floating-point values in an XMM register
VCVTPH2PS ymmreg,xmmrm128	Convert eight half-precision floating point values in memory or an XMM register (the bottom half of a YMM register) to eight single-precision floating-point values in a YMM register
VCVTPS2PH xmmrm64,xmmreg,imm8	Convert four single-precision floating point values in an XMM register to half-precision floating-point values in memory or the bottom half an XMM register; imm8 selects the rounding mode
VCVTPS2PH xmmrm128,ymmreg,imm8	Convert eight single-precision floating point values in a YMM register to half-precision floating-point values in memory or an XMM register; imm8 selects the rounding mode

NOTE: Support for these instructions is indicated by bit 29 of ECX after using CPUI with EAX=1

Interrupts

Standard ISA IRQs

- **0** - Programmable Interrupt Timer Interrupt
- **1** - Keyboard Interrupt
- **2** - Cascade (used internally by the two PICs; never raised)
- **3** - COM2 (if enabled)
- **4** - COM1 (if enabled)
- **5** - LPT2 (if enabled)
- **6** - Floppy Disk
- **7** - LPT1 / Unreliable "spurious" interrupt (usually)
- **8** - CMOS real-time clock (if enabled)
- **9** - Free for peripherals / legacy SCSI / NIC
- **10** - Free for peripherals / SCSI / NIC
- **11** - Free for peripherals / SCSI / NIC
- **12** - PS2 Mouse
- **13** - FPU / Coprocessor / Inter-processor
- **14** - Primary ATA Hard Disk
- **15** - Secondary ATA Hard Disk

Default PC Interrupt Vectors

- **0-31** - Protected Mode Exceptions (Reserved by Intel)
- **8-15** - Default mapping of IRQ0-IRQ7 by the BIOS at bootstrap
- **0x70-0x78** - Default mapping of IRQ8-IRQ15 by the BIOS at bootstrap

Hardware Exceptions

- **0x00** - Division by zero
- **0x01** - Debugger
- **0x02** - Non-Maskable interrupt (NMI)
- **0x03** - Breakpoint
- **0x04** - Overflow
- **0x05** - Bounds
- **0x06** - Invalid Opcode
- **0x07** - Coprocessor not available
- **0x08** - Double fault
- **0x09** - Coprocessor Segment Overrun (386 or earlier only)
- **0x0a** - Invalid Task State Segment
- **0x0b** - Segment not present
- **0x0c** - Stack Fault
- **0x0d** - General protection fault
- **0x0e** - Page Fault
- **0x0f** - Reserved
- **0x10** - Math Fault
- **0x11** - Alignment Check
- **0x12** - Machine Check
- **0x13** - SIMD Floating-Point Exception (extended math fault)
- **0x14** - Virtualization Exception
- **0x15-0x1f** - reserved

Created by Devyn Collier Johnson <DevynCJohnson@Gmail.com> (2016v2) More cheatsheets at DCJTech.info