P. Specht's "Liste der 8-Byte-Floatingpoint Befehle des masm32 Assemblers"

COMPACTED INTEL PENTIUM-4 PRESCOTT (April 2004) DPFP COMMAND SET

ADDPD	Add Packed Double-precision Floating-Point Values (8 byte)			
ADDSD	Add Iow Scalar Double-precision Floating-Point Values			
ADDSUBPD	: Packed Double-FP Add/Subtract in the high quadword of source			
	stores the result in the high quadword of the destination			
ANDPD	Bitwise Logical AND of Packed Double-precision Floating-Point Values			
ANDNPD	Bitwise Logical AND NOT of Packed Double-precision Floating-Point Values			
СМРРД	Compare Packed Double-precision Floating-Point values			
	EQ 0 000B Equal A=B			
	LE 2 010B Less-fnan-or-equal A≤B			
	UNORD 3 011B Unordered A,B = Unordered			
	NLT 5 101B Not-less-than NOT(A < B)			
	NLE 6 110B Not-less-than-oregul $NOT(A \le B)$			
CMPSD	Compare Scalar Double-precision Floating-Point Values			
	= compare dword at address DS:(E)SI with dword at address ES:(E)DI;			
COMISD	Compare Scalar Ordered Double-precision Floating-Point Values and Set EFLAGS			
	Compare low doubleprecision floating-point values in <i>xmm1</i> and <i>xmm2/mem64</i> , set the EFLAGS			
CVTDQ2PD	Convert Packed Doubleword Integers to Packed Double-precision Floating-Point			
CV/TRD 2D 0	from <i>xmm2/m128</i> to two packed double-precision floating-point values in <i>xmm1</i> .			
CVTPD2DQ	Convert Packed Double-precision Floating-Point values to Packed Doublewd Integers			
	<i>xmm2/m126</i> to two packed signed doubleword integers in <i>xmm1</i>			
	Convert Packed Double-precision Floating-Point Values to Packed Doublewd Integers			
	Convert Packed Double-precision Floating-Foint Values to Facked Single-Frecision Floating-Point			
	Convert Packed Single-Precision Floating-Point Values to Packed Double-precisionEP			
CVTSD2SI	Convert Scalar Double-precision Floating-Point Value to Doubleword Integer			
CVTSD2SS	Convert Scalar Double-precision Floating-Point Value to Scalar Single-PrecisionFP			
CVTSI2SD	Convert Doubleword Integer to Scalar Double-precision Floating-Point Value			
	Convert Scalar Single-Precision Floating-Point Value to Scalar Double-precision FP			
CVTSS2SD	Convert Scalar Single-Precision Floating-Point Value to Scalar Double-precision FP			
CVTSS2SD CVTTPD2PI	Convert Scalar Single-Precision Floating-Point Value to Scalar Double-precision FP Convert with Truncation Packed Double-precision FP to Packed Doubleword Integers			
CVTSS2SD CVTTPD2PI CVTTPD2DQ	Convert Scalar Single-Precision Floating-Point Value to Scalar Double-precision FP Convert with Truncation Packed Double-precision FP to Packed Doubleword Integers Convert with Truncation Packed Double-precision FP to Packed Doubleword Integers			
CVTSS2SD CVTTPD2PI CVTTPD2DQ CVTTSD2SI	Convert Scalar Single-Precision Floating-Point Value to Scalar Double-precision FP Convert with Truncation Packed Double-precision FP to Packed Doubleword Integers Convert with Truncation Packed Double-precision FP to Packed Doubleword Integers Convert with Truncation Scalar Double-precision FP to Signed Doubleword Integer			
CVTSS2SD CVTTPD2PI CVTTPD2DQ CVTTSD2SI DIVPD	Convert Scalar Single-Precision Floating-Point Value to Scalar Double-precision FP Convert with Truncation Packed Double-precision FP to Packed Doubleword Integers Convert with Truncation Packed Double-precision FP to Packed Doubleword Integers Convert with Truncation Scalar Double-precision FP to Signed Doubleword Integer Divide Packed Double-precision FP to Signed Doubleword Integer			
CVTSS2SD CVTTPD2PI CVTTPD2DQ CVTTSD2SI DIVPD	Convert Scalar Single-Precision Floating-Point Value to Scalar Double-precision FP Convert with Truncation Packed Double-precision FP to Packed Doubleword Integers Convert with Truncation Packed Double-precision FP to Packed Doubleword Integers Convert with Truncation Scalar Double-precision FP to Signed Doubleword Integer Divide Packed Double-precision FP to Signed Doubleword Integer Divide Packed Double-precision FP to Signed Doubleword Integer in xmm1 by packed double-precision floating-point Values xmm2/m128.			
CVTSS2SD CVTTPD2PI CVTTPD2DQ CVTTSD2SI DIVPD DIVSD	Convert Scalar Single-Precision Floating-Point Value to Scalar Double-precision FP Convert with Truncation Packed Double-precision FP to Packed Doubleword Integers Convert with Truncation Packed Double-precision FP to Packed Doubleword Integers Convert with Truncation Scalar Double-precision FP to Signed Doubleword Integer Divide Packed Double-precision Floating-Point Values in xmm1 by packed doubleprecision floating-point values xmm2/m128. Divide Scalar Double-precision Floating-Point Values			
CVTSS2SD CVTTPD2PI CVTTPD2DQ CVTTSD2SI DIVPD DIVSD	Convert Scalar Single-Precision Floating-Point Value to Scalar Double-precision FP Convert with Truncation Packed Double-precision FP to Packed Doubleword Integers Convert with Truncation Packed Double-precision FP to Packed Doubleword Integers Convert with Truncation Scalar Double-precision FP to Signed Doubleword Integers Divide Packed Double-precision Floating-Point Values in xmm1 by packed doubleprecision floating-point values xmm2/m128. Divide Scalar Double-precision Floating-Point Values Divide low double-precision floating-point value in xmm1 by low double-precision Point Values Divide low double-precision floating-point value in xmm1 by low double-precision			
CVTSS2SD CVTTPD2PI CVTTPD2DQ CVTTSD2SI DIVPD DIVSD	Convert Scalar Single-Precision Floating-Point Value to Scalar Double-precision FP Convert with Truncation Packed Double-precision FP to Packed Doubleword Integers Convert with Truncation Scalar Double-precision FP to Packed Doubleword Integers Convert with Truncation Scalar Double-precision FP to Signed Doubleword Integers Divide Packed Double-precision Floating-Point Values in xmm1 by packed double-precision floating-point values xmm2/m128. Divide Scalar Double-precision Floating-Point Values Divide low double-precision floating-point value in xmm1 by low double-precision floating-point value in xmm1 by low double-precision floating-point value in xmm1 by low double-precision Convert value in xmm2/mem64.			
CVTSS2SD CVTTPD2PI CVTTPD2DQ CVTTSD2SI DIVPD DIVSD	Convert Scalar Single-Precision Floating-Point Value to Scalar Double-precision FP Convert with Truncation Packed Double-precision FP to Packed Doubleword Integers Convert with Truncation Packed Double-precision FP to Packed Doubleword Integers Convert with Truncation Scalar Double-precision FP to Signed Doubleword Integers Divide Packed Double-precision Floating-Point Values in xmm1 by packed double-precision floating-point values xmm2/m128. Divide Scalar Double-precision Floating-Point Values Divide low double-precision floating-point value in xmm1 by low double-precision floating-point value in xmm1 by low double-precision floating-point value in xmm2/mem64. Compute 2x-1			
CVTSS2SD CVTTPD2PI CVTTPD2DQ CVTTSD2SI DIVPD DIVSD F2XM1 FABS EADD	Convert Scalar Single-Precision Floating-Point Value to Scalar Double-precision FP Convert with Truncation Packed Double-precision FP to Packed Doubleword Integers Convert with Truncation Scalar Double-precision FP to Packed Doubleword Integers Convert with Truncation Scalar Double-precision FP to Signed Doubleword Integers Divide Packed Double-precision Floating-Point Values in xmm1 by packed double-precision floating-point values xmm2/m128. Divide Scalar Double-precision Floating-Point Values Divide low double-precision floating-point value in xmm1 by low double-precision floating-point value in xmm1 by low double-precision floating-point value in xmm2/mem64. Compute 2x-1 Absolute Value			
CVTSS2SD CVTTPD2PI CVTTPD2DQ CVTTSD2SI DIVPD DIVSD F2XM1 FABS FADD	Convert Scalar Single-Precision Floating-Point Value to Scalar Double-precision FP Convert with Truncation Packed Double-precision FP to Packed Doubleword Integers Convert with Truncation Scalar Double-precision FP to Packed Doubleword Integers Convert with Truncation Scalar Double-precision FP to Signed Doubleword Integers Divide Packed Double-precision Floating-Point Values in xmm1 by packed doubleprecision floating-point values xmm2/m128. Divide Scalar Double-precision Floating-Point Values Divide low double-precision floating-point value in xmm1 by low double-precision floating-point value in xmm1 by low double-precision floating-point value in xmm2/mem64. Compute 2x-1 Absolute Value FADD m32fp			
CVTSS2SD CVTTPD2PI CVTTPD2DQ CVTTSD2SI DIVPD DIVSD F2XM1 FABS FADD	Convert Scalar Single-Precision Floating-Point Value to Scalar Double-precision FP Convert with Truncation Packed Double-precision FP to Packed Doubleword Integers Convert with Truncation Scalar Double-precision FP to Signed Doubleword Integers Convert with Truncation Scalar Double-precision FP to Signed Doubleword Integers Divide Packed Double-precision Floating-Point Values in xmm1 by packed doubleprecision floating-point values xmm2/m128. Divide Scalar Double-precision Floating-Point Values Divide low double-precision floating-point value in xmm1 by low double-precision floating-point value in xmm1 by low double-precision floating-point value in xmm2/mem64. Compute 2x-1 Absolute Value FADD m32fp FADD m64fp FADD ST(0), ST(i)			
CVTSS2SD CVTTPD2PI CVTTPD2DQ CVTTSD2SI DIVPD DIVSD F2XM1 FABS FADD	Convert Scalar Single-Precision Floating-Point Value to Scalar Double-precision FP Convert with Truncation Packed Double-precision FP to Packed Doubleword Integers Convert with Truncation Scalar Double-precision FP to Signed Doubleword Integers Convert with Truncation Scalar Double-precision FP to Signed Doubleword Integers Divide Packed Double-precision Floating-Point Values in xmm1 by packed double-precision floating-point values xmm2/m128. Divide Scalar Double-precision floating-Point Values Divide low double-precision floating-point value in xmm1 by low double-precision floating-point value in xmm1 by low double-precision floating-point value in xmm1 by low double-precision floating-point value in xmm2/mem64. Compute 2x-1 Absolute Value FADD m32fp FADD ST(0), ST(i) FADD ST(0), ST(i)			
CVTSS2SD CVTTPD2PI CVTTPD2DQ CVTTSD2SI DIVPD DIVSD F2XM1 FABS FADD FADDP	Convert Scalar Single-Precision Floating-Point Value to Scalar Double-precision FP Convert with Truncation Packed Double-precision FP to Packed Doubleword Integers Convert with Truncation Scalar Double-precision FP to Signed Doubleword Integers Divide Packed Double-precision Floating-Point Values in xmm1 by packed double-precision floating-point values xmm2/m128. Divide Scalar Double-precision Floating-Point Values Divide Scalar Double-precision Floating-Point Values Divide low double-precision floating-point value in xmm1 by low double-precision floating-point value in xmm2/mem64. Compute 2x-1 Absolute Value FADD m64fp FADD ST(0), ST(i) FADD ST(i), ST(0) FADDP ST(i), ST(0)			
CVTSS2SD CVTTPD2PI CVTTPD2DQ CVTTSD2SI DIVPD DIVSD F2XM1 FABS FADD FIADD	Convert Scalar Single-Precision Floating-Point Value to Scalar Double-precision FP Convert with Truncation Packed Double-precision FP to Packed Doubleword Integers Convert with Truncation Scalar Double-precision FP to Signed Doubleword Integers Convert with Truncation Scalar Double-precision FP to Signed Doubleword Integers Divide Packed Double-precision Floating-Point Values in xmm1 by packed doubleprecision floating-point values xmm2/m128. Divide Scalar Double-precision Floating-Point Values Divide low double-precision floating-point value in xmm1 by low double-precision floating-point value in xmm2/m128. Compute 2x-1 Absolute Value FADD m32fp FADD m64fp FADD ST(0), ST(i) FADD ST(i), ST(0) FADDP Add ST(0) to ST(1), store result in ST(1), and pop the register stack.			
CVTSS2SD CVTTPD2PI CVTTPD2DQ CVTTSD2SI DIVPD DIVSD F2XM1 FABS FADD FIADD	Convert Scalar Single-Precision Floating-Point Value to Scalar Double-precision FP Convert with Truncation Packed Double-precision FP to Packed Doubleword Integers Convert with Truncation Scalar Double-precision FP to Signed Doubleword Integers Convert with Truncation Scalar Double-precision FP to Signed Doubleword Integers Divide Packed Double-precision Floating-Point Values in xmm1 by packed double-precision floating-point values xmm2/m128. Divide Scalar Double-precision floating-Point Values Divide low double-precision floating-point value in xmm1 by low double-precision floating-point value in xmm2/mem64. Compute 2x–1 Absolute Value FADD m64fp FADD ST(0), ST(0) FADDP Add ST(0) to ST(1), store result in ST(1), and pop the register stack. FIADD m32int			
CVTSS2SD CVTTPD2PI CVTTPD2DQ CVTTSD2SI DIVPD DIVSD F2XM1 FABS FADD FIADD	Convert Scalar Single-Precision Floating-Point Value to Scalar Double-precision FP Convert with Truncation Packed Double-precision FP to Packed Doubleword Integers Convert with Truncation Scalar Double-precision FP to Signed Doubleword Integers Divide Packed Double-precision Floating-Point Values in xmm1 by packed doubleprecision floating-point values xmm2/m128. Divide Scalar Double-precision floating-Point Values Divide low double-precision floating-point value in xmm1 by flow floating-point value in xmm1 by floating-p			
CVTSS2SD CVTTPD2PI CVTTPD2DQ CVTTSD2SI DIVPD DIVSD F2XM1 FABS FADD FIADD FIADD	Convert Scalar Single-Precision Floating-Point Value to Scalar Double-precision FP Convert with Truncation Packed Double-precision FP to Packed Doubleword Integers Convert with Truncation Scalar Double-precision FP to Signed Doubleword Integer Divide Packed Double-precision Floating-Point Values in xmm1 by packed double-precision Floating-Point Values Divide Scalar Double-precision Floating-Point Values Divide Scalar Double-precision Floating-Point Values Divide Scalar Double-precision Floating-Point Values Divide low double-precision floating-point value in xmm1 by low double-precision floating-point value in xmm1 by low double-precision floating-point value in xmm1 by low double-precision floating-point value in xmm1 by low double-precision floating-point value in xmm1 by low double-precision floating-point value in xmm1 by low double-precision floating-point value FADD m32/p FADD m64/p FADD ST(i), ST(0) FADDP Add ST(0) to ST(1), store result in ST(1), and pop the register stack. FIADD m32int FIADD m16int Change Sign			
CVTSS2SD CVTTPD2PI CVTTPD2DQ CVTTSD2SI DIVPD DIVSD F2XM1 FABS FADD FIADD FIADD FIADD FCHS FCMOVcc	Convert Scalar Single-Precision Floating-Point Value to Scalar Double-precision FP Convert with Truncation Packed Double-precision FP to Packed Doubleword Integers Convert with Truncation Scalar Double-precision FP to Signed Doubleword Integers Divide Packed Double-precision Floating-Point Values in xmm1 by packed double-precision floating-point values xmm2/m128. Divide Scalar Double-precision floating-Point Values Divide Scalar Double-precision floating-point values xmm2/m128. Divide low double-precision floating-point value in xmm1 by low double-precision floating-point value in xmm1 by acked floating-point value in xmm1 by low double-precision floating-point value in xmm2/mem64. Compute 2x-1 Absolute Value FADD m32(i) FADD m32(i) FADD m32(i) FADD m32(i) FADD m32(i) FADD m3			
CVTSS2SD CVTTPD2PI CVTTPD2DQ CVTTSD2SI DIVPD DIVSD F2XM1 FABS FADD FIADD FIADD FCHS FCHS FCMOVcc	Convert Scalar Single-Precision Floating-Point Value to Scalar Double-precision FP Convert with Truncation Packed Double-precision FP to Packed Doubleword Integers Convert with Truncation Packed Double-precision FP to Signed Doubleword Integers Convert with Truncation Packed Double-precision FP to Signed Doubleword Integers Convert with Truncation Scalar Double-precision FP to Signed Doubleword Integer Divide Packed Double-precision Floating-Point Values in xmm1 by packed double-precision Floating-Point Values Divide Scalar Double-precision Floating-Point Values Divide Scalar Double-precision Floating-Point Values Divide Scalar Double-precision Floating-Point Values Divide Iso double-precision floating-point values xmm2/m128. Divide Iso double-precision floating-point Values Divide Iso Modify FADD m32/p FADD modify FADD ST(0), ST(0) FADD Modify			
CVTSS2SD CVTTPD2PI CVTTPD2DQ CVTTSD2SI DIVPD DIVSD F2XM1 FABS FADD FIADD FIADD FCHS FCMOVcc	Convert Scalar Single-Precision Floating-Point Value to Scalar Double-precision FP Convert with Truncation Packed Double-precision FP to Packed Doubleword Integers Convert with Truncation Scalar Double-precision FP to Signed Doubleword Integers Convert with Truncation Scalar Double-precision FP to Signed Doubleword Integers Divide Packed Double-precision Floating-Point Values in xmm1 by packed double-precision floating-point values xmm2/m128. Divide Scalar Double-precision floating-point values xmm1 by low double-precision floating-point value in xmm1 by floating-point value in xmm1 by low double-precision floating-point value in xmm1 by floating the structure in xmm1 by floating-point v			
CVTSS2SD CVTTPD2PI CVTTPD2DQ CVTTSD2SI DIVPD DIVSD F2XM1 FABS FADD FIADD FIADD FCHS FCMOVcc	Convert Scalar Single-Precision Floating-Point Value to Scalar Double-precision FP Convert with Truncation Packed Double-precision FP to Packed Doubleword Integers Convert with Truncation Scalar Double-precision FP to Signed Doubleword Integers Divide Packed Double-precision Floating-Point Values in xmm1 by packed doubleprecision floating-point values xmm2/m128. Divide Scalar Double-precision floating-point values xmm2/m128. Divide Scalar Double-precision floating-point values xmm1 by low double-precision floating-point value in xmm1 by float value FADD m32fp FADD m32int FIADD m32int FIADD m32int FIADD m32int FIADD m32int FIADD m32int FCMOVB ST(0, ST(0) Move if below (CF=1)			
CVTSS2SD CVTTPD2PI CVTTPD2DQ CVTTSD2SI DIVPD DIVSD F2XM1 FABS FADD FIADD FIADD FCHS FCMOVcc	Convert Scalar Single-Precision Floating-Point Value to Scalar Double-precision FP Convert with Truncation Packed Double-precision FP to Packed Doubleword Integers Convert with Truncation Scalar Double-precision FP to Signed Doubleword Integers Divide Packed Double-precision Floating-Point Values in xmm1 by packed double-precision floating-Point Values in xmm1 by packed double-precision floating-Point Values Divide Scalar Double-precision floating-point value in xmm1 by low double-precision floating-point value in xmm1 by low double-precision Compute 2x-1			
CVTSS2SD CVTTPD2PI CVTTPD2DQ CVTTSD2SI DIVPD DIVSD F2XM1 FABS FADD FIADD FCHS FCMOVcc	Convert Scalar Single-Precision Floating-Point Value to Scalar Double-precision FP Convert with Truncation Packed Double-precision FP to Packed Doubleword Integers Convert with Truncation Scalar Double-precision FP to Signed Doubleword Integers Divide Packed Double-precision Floating-Point Values in xmm1 by packed double-precision floating-point values xmm2/m128. Divide Scalar Double-precision floating-point Values Divide Scalar Double-precision floating-point values xmm2/m128. Divide low double-precision floating-point value in xmm1 by low double-precision floating-point value in xmm2/mem64. Compute 2x-1 Absolute Value FADD m32fp FADD ST(i), ST(i) FADD ST(i), ST(i) FADD M64fp FADD ST(i), ST(i) FADD M32fp FADD M64fp FADD ST(i), ST(i) FADD M32fp FADD M64fp FADD M32fp FADD M64fp FADD M32fp FADD M32fp FADD M64fp FADD M32fp			
CVTSS2SD CVTTPD2PI CVTTPD2DQ CVTTSD2SI DIVPD DIVSD F2XM1 FABS FADD FIADD FCHS FCMOVcc	Convert Scalar Single-Precision Floating-Point Value to Scalar Double-precision FP Convert with Truncation Packed Double-precision FP to Packed Doubleword Integers Convert with Truncation Scalar Double-precision FP to Signed Doubleword Integers Divide Packed Double-precision FP to Signed Doubleword Integers Divide Packed Double-precision FP to Signed Doubleword Integers Divide Packed Double-precision Floating-Point Values in xmm1 by packed double-precision floating-point values xmm2/m128. Divide Scalar Double-precision floating-point Values Divide low double-precision floating-point value in xmm1 by low double-precision floating-point value in xmm2/mem64. Compute 2x-1 Absolute Value FADD m32/p FADD m64fp FADD ST(i), ST(0) FADDP Add ST(0) to ST(1), store result in ST(1), and pop the register stack. FIADD m32int FIADD m32int FIADD m16int Change Sign FCMOVB ST(0), ST(0) FCMOVB ST(0), ST(0) FCMOVB ST(0), ST(0) FCMOVB ST(0), ST(0) Move if below (CF=1) FCMOVNB ST(0), ST(0) Move if not sequal (CF=1) or ZF=1) FCMOVNB ST(0), ST(0) Move if not below (CF=1) FCMOVNB ST(0), ST(0) Mo			

FCOS	Cosine
FDIV	Divide
FDIVP	
FIDIV	
FDIVR	Reverse Divide
FDIVRP	
	Free Fleeting Doint Pogister
ETNIT	Initialize EDI Lafter obsolving for pending upmasked fleating point exceptions
FNINIT	Initialize FPO alter checking for pending unmasked fleating point exceptions.
FLD	Load Electing Point Value
	Luch m32fp onto the EDL register stack
	Push $m52jp$ onto the FDI register stack.
	Dush $m80fn$ onto the EDLI register stack.
	Push ST(i) onto the FPU register stack.
FLD1	Load Constant
FLDL2T	
FLDL2E	
FLDPI	
FLDLG2	
	Multiply
FMULP	
FIMUL	
FNOP	No Operation
FPATAN	Partial Arctangent
FPREM	Partial Remainder
FPREM1	Partial Remainder
FPTAN	Partial Tangent
FRNDINT	Round to Integer
FSCALE	Scale
FSIN	Sine
FSINCOS	Sine and Cosine
FSQRT	Square Root
FST	Store Floating Point Value
FSTP	Copy ST(0) to $m32fp$.
	Copy ST(0) to $m64fp$.
	Copy ST(0) to ST(i).
	Copy ST(0) to <i>m32fp</i> and pop register stack.
	Copy ST(0) to <i>m64fp</i> and pop register stack.
	Copy S1(0) to m Vite and non-register stack
	copy S1(0) to mooth all pop register stack.
	Copy ST(0) to ST(i) and pop register stack.
FSUB	Copy ST(0) to ST(i) and pop register stack. Subtract
FSUB FSUBP FISUB	Copy ST(0) to ST(i) and pop register stack. Subtract
FSUB FSUBP FISUB FSUBR	Copy ST(0) to <i>moop</i> and pop register stack. Copy ST(0) to ST(i) and pop register stack. Subtract
FSUB FSUBP FISUB FSUBR FSUBRP	Copy ST(0) to ST(i) and pop register stack. Subtract Reverse Subtract
FSUB FSUBP FISUB FSUBR FSUBRP FISUBR	Copy ST(0) to ST(i) and pop register stack. Subtract Reverse Subtract
FSUB FSUBP FISUB FSUBR FSUBRP FISUBR FTST	Copy ST(0) to ST(i) and pop register stack. Copy ST(0) to ST(i) and pop register stack. Subtract Reverse Subtract TEST
FSUB FSUBP FISUB FSUBR FSUBRP FISUBR FTST	Copy ST(0) to ST(i) and pop register stack. Subtract TEST Unordered Compare Floating Point Values
FSUB FSUBP FISUB FSUBR FSUBRP FISUBR FTST FUCOM	Copy ST(0) to ST(i) and pop register stack. Copy ST(0) to ST(i) and pop register stack. Subtract TEST Unordered Compare Floating Point Values Compare ST(0) with ST(i). Owner ST(0) with ST(i).
FSUB FSUBP FISUB FSUBR FSUBRP FISUBR FTST FUCOM	Copy ST(0) to ST(i) and pop register stack. Copy ST(0) to ST(i) and pop register stack. Subtract TEST Unordered Compare Floating Point Values Compare ST(0) with ST(i). Compare ST(0) with ST(1). Compare ST(0) with ST(1).
FSUB FSUBP FISUB FSUBR FSUBR FISUBR FTST FUCOM FUCOMP FUCOMPP	Copy ST(0) to ST(i) and pop register stack. Copy ST(0) to ST(i) and pop register stack. Subtract TEST Unordered Compare Floating Point Values Compare ST(0) with ST(i). Compare ST(0) with ST(1). Compare ST(0) with ST(i) and pop register stack. Compare ST(0) with ST(i). Compare ST(0) with ST(i). Unordered Compare ST(0) with ST(i).
FSUB FSUBP FSUBR FSUBR FSUBR FISUBR FTST FUCOM FUCOMP FUCOMPP	Copy ST(0) to ST(i) and pop register stack. Copy ST(0) to ST(i) and pop register stack. Subtract TEST Unordered Compare Floating Point Values Compare ST(0) with ST(i). Compare ST(0) with ST(i). Compare ST(0) with ST(i). Compare ST(0) with ST(i) and pop register stack.
FSUB FSUBP FISUBR FSUBR FSUBR FISUBR FTST FUCOM FUCOMP FUCOMPP	Copy ST(0) to ST(i) and pop register stack. Copy ST(0) to ST(i) and pop register stack. Subtract TEST Unordered Compare Floating Point Values Compare ST(0) with ST(i). Compare ST(0) with ST(1). Compare ST(0) with ST(i) and pop register stack. Compare ST(0) with ST(i) and pop register stack. Compare ST(0) with ST(1) and pop register stack.
FSUB FSUBP FISUB FSUBR FSUBR FISUBR FTST FUCOM FUCOMP FUCOMPP FUCOMPP	Copy ST(0) to ST(i) and pop register stack. Copy ST(0) to ST(i) and pop register stack. Subtract TEST Unordered Compare Floating Point Values Compare ST(0) with ST(i). Compare ST(0) with ST(1). Compare ST(0) with ST(i) and pop register stack. Compare ST(0) with ST(1) and pop register stack twice. ExamineModR/M
FSUB FSUBP FISUBR FSUBR FSUBR FISUBR FTST FUCOM FUCOMP FUCOMPP FUCOMPP	Copy ST(0) to ST(i) and pop register stack. Subtract Reverse Subtract TEST Unordered Compare Floating Point Values Compare ST(0) with ST(i). Compare ST(0) with ST(1). Compare ST(0) with ST(i) and pop register stack. Compare ST(0) with ST(1) and pop register stack. Description ST(0) with ST(1) and pop register stack twice. ExamineModR/M Exchange Register Contents Poetor x97 EBUL MMY Technology, SSE SSE2, and SSE2 State
FSUB FSUBP FISUB FSUBR FSUBRP FISUBR FTST FUCOM FUCOMP FUCOMPP FUCOMPP FXAM FXCH FXRSTOR EYSAVE	Copy ST(0) to ST(i) and pop register stack. Copy ST(0) to ST(i) and pop register stack. Subtract TEST Unordered Compare Floating Point Values Compare ST(0) with ST(i). Compare ST(0) with ST(1). Compare ST(0) with ST(i) and pop register stack. Compare ST(0) with ST(1) and pop register stack twice. ExamineModR/M Exchange Register Contents Restore x87 FPU, MMX Technology, SSE, and SSE3 State Save v87 FPU, MMX Technology, SSE and SSE3 State
FSUB FSUBP FISUBR FSUBR FSUBR FISUBR FTST FUCOM FUCOMP FUCOMPP FUCOMPP FUCOMPP FUCOMPP FXAM FXCH FXRSTOR FXSAVE FXSAVE	Copy ST(0) to <i>missip</i> and pop register stack. Copy ST(0) to ST(i) and pop register stack. Subtract TEST Unordered Compare Floating Point Values Compare ST(0) with ST(i). Compare ST(0) with ST(1). Compare ST(0) with ST(i) and pop register stack. Compare ST(0) with ST(1) and pop register stack twice. ExamineModR/M Exchange Register Contents Restore x87 FPU, MMX Technology, SSE, and SSE3 State Save x87 FPU, MMX Technology, SSE, and SSE2 State
FSUB FSUBP FISUBR FSUBR FSUBR FISUBR FTST FUCOM FUCOMP FUCOMPP FUCOMPP FUCOMPP FXCM FXCH FXRSTOR FXSAVE FXTRACT FYL2X	Copy ST(0) to ST(i) and pop register stack. Subtract Reverse Subtract Unordered Compare Floating Point Values Compare ST(0) with ST(i). Compare ST(0) with ST(1). Compare ST(0) with ST(i). Compare ST(0) with ST(i). Compare ST(0) with ST(i). Compare ST(0) with ST(i) and pop register stack. Compare ST(0) with ST(1) and pop register stack.
FSUB FSUBP FISUBR FSUBR FSUBR FSUBR FTST FUCOM FUCOMP FUCOMPP FUCOMPP FUCOMPP FXCM FXCH FXRSTOR FXSAVE FXTRACT FYL2X FYL2XP1	Copy ST(0) to ST(i) and pop register stack. Subtract Image: Provide the stack of the sta
FSUB FSUBP FISUBR FSUBR FSUBR FISUBR FTST FUCOM FUCOMP FUCOMPP FUCOMPP FUCOMPP FXCM FXCH FXRSTOR FXSAVE FXTRACT FYL2X FYL2XP1 HADDPD	Copy ST(0) to ST(i) and pop register stack. Subtract TEST Unordered Compare Floating Point Values Compare ST(0) with ST(i). Compare ST(0) with ST(1). Compare ST(0) with ST(i) and pop register stack. Compare ST(0) with ST(1) and pop register stack. Compare X87 FPU, MMX Technology, SSE, SSE2, and SSE3 State Save x87 FPU, MMX Technology, SSE, and SSE2 State Extract Exponent and Significand Co

	[127:64] [63	3:0] xmm2 /m128			
	[127:64] [53	3:0] xmm1			
	·	_			
	xmm2/m128[83:0] + xmm2/m128[127:64] xmm1[63:0] + :	xmm1[127:64] Result: xmm1			
HSUBPD	: Packed Double-FP Horizon	tal Subtract			
	like HADDPD, but subtract from lower the upper				
Jcc	Jump if Condition Is Met *)				
ЈМР	Jump				
LDDQU	: Load Unaligned Integer 128	8 Bits			
LEA	Load Effective Address				
LOOP	Loop According to ECX Cou	Inter			
LOOPCC	E2 cb LOOP rel8 Decrement	t count; jump short	if count ≠ 0.		
	E1 <i>cb</i> LOOPE <i>rel8</i> Decrement count; jump short if count \neq 0 and ZF = 1.				
	E0 cb LOOPNE rel8 Decrem	nent count; jump sh	ort if count ≠ 0	and $ZF = 0$.	
MASKMOVDQU	Store Selected Bytes of Doul	ble Quadword			
	Selectively write bytes from xm	<i>nm1</i> to memory loca	tion using the I	byte mask in <i>xn</i>	<i>m2</i> . The default
	memory location is specified by	y DS:EDI.			
MASKMOVQ	Store Selected Bytes of Qua	dword			
MAXPD	Return Maximum Packed Do	ouble-precision Flo	oating-Point V	alues	
	Return the maximum double-p	precision floating-po	int values betw	een xmm2/m12/	8 and <i>xmm1</i> .
	The most significant bit in each by	yte of the mask opera	ind determines v	whether the corre	sponding byte in the
	source operand is written to the co	orresponding byte loc	ation in memory	y: 0 indicates no	write and 1 indicates
	write.	··· ·· -·		-	
MAXSD	Return Maximum Scalar Dou	ble-precision Floa	ating-Point Va	lue	
	Return the maximum scalar do	publeprecision floati	ng-point value	between xmm2	/mem64 and xmm1
MINPD	Return Minimum Packed Dou	uble-precision Flo	ating-Point Va	alues (see MA	XPD)
MINSD	Return Minimum Scalar Doul	ible-precision Floa	ting-Point Va	lue (see MAXS	(U)
MOV	MOV r/m8,r8	Move r§	to	r/m8.	
	MOV r/m16,r16	Move r16	to	r/m16.	
	MOV r/m32,r32	Move r32	to	r/m32.	
	MOV <i>r8,r/m8</i>	Move r/mg	to	r8.	
	MOV r16,r/m16	Move r/ml6	to	r16.	
	MOV r32,r/m32	Move r/m32	to	r32.	
	MOV 7/m10,Sreg	Move segment	register to	r/m/g.	
	MOV AL moffs 8*	Nove hyte	to segmt	register to Al	
	MOV AX moffs 16*	Move word	at (seg. offset)	to AX	
	MOV FAX moffs 22*	Move doubleword	at(seg.offset)	to FAX	
	MOV moffs8 Al	Move Al	to(seg:offset)	10 27 01	
	MOV moffs16*.AX	Move AX	to(seg:offset).		
	MOV moffs32*,EAX	Move EAX	to(seg:offset).		
	MOV r8, imm8	Move imm8	to	r8.	
	MOV r16, imm16	Move imm16	to	r16.	
	MOV r32, imm32	Move imm32	to	r32.	
	MOV r/m8, imm8	Move imm8	to	r/m8.	
	MOV 7/m10, imm10	Nove immig	to	r/m10.	
		wove mm24	10	r/m24.	
MOVAPD	Move Aligned Packed Double	le-precision Floatii	ng-Point Value	es:	
	Move packed double-precision	n floating-point value	es from xmm2/m	n128 to xmm1.	
	Move packed double-precision	n floating-point value	es from <i>xmm1</i> t	o <i>xmm2/m128</i>	
MOVD	Move Doubleword				
ΜΟΥϘ	Move Quadword				
MOVDDUP	: Move One Double-FP and D	Duplicate			
	Move (lower) double-precision	floatingpoint value	from the lower	64-bit operand	in xmm2/m64 to xmm1
	and duplicate to the upper 64 k	bit of xmm1			
MOVDQA	Move Aligned Double Quadw	word			
MOVDQU	Move Unaligned Double Qua	adword	<u> </u>		
MOVDQ2Q	Move low Quadword from X	MM to MMX Techn	ology Registe	er	
MOVHPD	Move High64bit of Packed-D	ouble-precision F	loating-Point	Value	

	Move double-precision floating-point value from <i>m64</i> to high guadword of <i>xmm</i> .		
	Move double-precision floating-point value from high guadword of <i>xmm</i> to <i>m64</i> .		
MOVLPD	Move Low Packed Double-precision Floating-Point Value (see MOVHPD)		
MOVMSKPD	Extract Packed Double-precision Floating-Point Sign Mask		
	Extract 2-bit sign mask from <i>xmm</i> and store in r_{32} .		
MOVNTDO	Store Double Quadword Using Non-Temporal Hint		
-	Move double guadword from xmm to m128 using non-temporal hint.		
MOVNTPD	Store Packed Double-precision Floating-Point Values Using Non-Temporal Hint		
MOVO	Move Quadword		
MOVO2DO	Move Quadword from MMX Technology to low guadword of XMM Register		
MOVSD	Move Scalar Double-precision Floating-Point Value		
	Move scalar double-precision floating-point value from $xmm^2/m64$ to xmm^1 register.		
	Move scalar double-precision floating-point value from xmm1 register to xmm2/m64		
MOVUPD	Move Unaligned Packed Double-precision Floating-Point Values		
	Move packed double-precision floating-point values from $xmm^{2/m}128$ to xmm1		
	Move packed double-precision floating-point values from xmm1 to xmm2/m128.		
MOVZX	Move with Zero-Extend		
MULPD	Multiply Packed Double-precision Floating-Point Values in xmm2/m128 by xmm1		
MULSD	Multiply Scalar Double-precision Floating-Point Values		
	Multiply the low double-precision floating-point value in xmm2/mem64 by low double-precision floating-		
	point value in <i>xmm1</i> .		
ORPD	Bitwise Logical OR of Double-precision Floating-Point Values in <i>xmm2/m128</i> and <i>xmm1</i> .		
PAND	Logical AND		
	Bitwise AND mm/m64 and mm.		
	Bitwise AND of <i>xmm2/m128</i> and <i>xmm1</i> .		
	The destination operand can be an MMX technology register or an XMM register.		
PANDN	Logical AND NOT (see above)		
PAUSE	Spin Loop Hint		
PAVGB	Average Packed Integers		
PAVGW			
PCMPEQB	Compare Packed Data for Equal		
PCMPEQW			
FUPFLUD	Compare packed doublewords in <i>mm/m64</i> and <i>mmx</i> for equality		
	Compare packed doublewords in <i>mm/m64</i> and <i>mmx</i> for equality		
	Compare packed doublewords in <i>mm/m64</i> and <i>mmx</i> for equality Compare packed doublewords in <i>xmm2/m128</i> and xmm1 for equality		
PMOVMSKB	Compare packed doublewords in <i>mm/m64</i> and <i>mmx</i> for equality Compare packed doublewords in <i>xmm2/m128</i> and xmm1 for equality Move Byte Mask of <i>xmm or mmx</i> to <i>r32</i> .		
PMOVMSKB POP	Compare packed doublewords in <i>mm/m64</i> and <i>mmx</i> for equality Compare packed doublewords in <i>xmm2/m128</i> and xmm1 for equality Move Byte Mask of <i>xmm or mmx</i> to <i>r32</i> . Pop a Value from the Stack		
PMOVMSKB POP PSLLDQ	Compare packed doublewords in <i>mm/m64</i> and <i>mmx</i> for equality Compare packed doublewords in <i>xmm2/m128</i> and xmm1 for equality Move Byte Mask of <i>xmm or mmx</i> to <i>r32</i> . Pop a Value from the Stack Shift <i>xmm1</i> Double Quadword Left Logical by <i>imm8</i> bytes and by shifting in 0s.		
PMOVMSKB POP PSLLDQ PSRLDQ	Compare packed doublewords in <i>mm/m64</i> and <i>mmx</i> for equality Compare packed doublewords in <i>xmm2/m128</i> and xmm1 for equality Move Byte Mask of <i>xmm or mmx</i> to <i>r32</i> . Pop a Value from the Stack Shift <i>xmm1</i> Double Quadword Left Logical by <i>imm8</i> bytes and by shifting in 0s. Shift Double Quadword Right Logical Dueb Ward or Doubleword Orte the Stack		
PMOVMSKB POP PSLLDQ PSRLDQ PUSH	Compare packed doublewords in <i>mm/m64</i> and <i>mmx</i> for equality Compare packed doublewords in <i>xmm2/m128</i> and xmm1 for equality Move Byte Mask of <i>xmm or mmx</i> to <i>r32</i> . Pop a Value from the Stack Shift <i>xmm1</i> Double Quadword Left Logical by <i>imm8</i> bytes and by shifting in 0s. Shift Double Quadword Right Logical Push Word or Doubleword Onto the Stack Dueb EEL ACS Paginter onto the Stack		
PMOVMSKB POP PSLLDQ PSRLDQ PUSH PUSHF PUSHFD	Compare packed doublewords in mm/m64 and mmx for equality Compare packed doublewords in xmm2/m128 and xmm1 for equality Move Byte Mask of xmm or mmx to r32. Pop a Value from the Stack Shift xmm1 Double Quadword Left Logical by imm8 bytes and by shifting in 0s. Shift Double Quadword Right Logical Push Word or Doubleword Onto the Stack Push EFLAGS Register onto the Stack		
PMOVMSKB POP PSLLDQ PSRLDQ PUSHF PUSHF PUSHFD PXOR	Compare packed doublewords in <i>mm/m64</i> and <i>mmx</i> for equality Compare packed doublewords in <i>xmm2/m128</i> and xmm1 for equality Move Byte Mask of <i>xmm or mmx</i> to <i>r32</i> . Pop a Value from the Stack Shift <i>xmm1</i> Double Quadword Left Logical by <i>imm8</i> bytes and by shifting in 0s. Shift Double Quadword Right Logical Push Word or Doubleword Onto the Stack Push EFLAGS Register onto the Stack Logical Exclusive OR		
PMOVMSKB POP PSLLDQ PSRLDQ PUSHF PUSHF PUSHFD PXOR REP	Compare packed doublewords in <i>mm/m64</i> and <i>mmx</i> for equality Compare packed doublewords in <i>xmm2/m128</i> and xmm1 for equality Move Byte Mask of <i>xmm or mmx</i> to <i>r32</i> . Pop a Value from the Stack Shift <i>xmm1</i> Double Quadword Left Logical by <i>imm8</i> bytes and by shifting in 0s. Shift Double Quadword Right Logical Push Word or Doubleword Onto the Stack Push EFLAGS Register onto the Stack Logical Exclusive OR Repeat String Operation Prefix		
PMOVMSKB POP PSLLDQ PSRLDQ PUSHF PUSHFD PUSHFD PXOR REP REPE	Compare packed doublewords in <i>mm/m64</i> and <i>mmx</i> for equality Compare packed doublewords in <i>xmm2/m128</i> and xmm1 for equality Move Byte Mask of <i>xmm or mmx</i> to <i>r32</i> . Pop a Value from the Stack Shift <i>xmm1</i> Double Quadword Left Logical by <i>imm8</i> bytes and by shifting in 0s. Shift Double Quadword Right Logical Push Word or Doubleword Onto the Stack Push EFLAGS Register onto the Stack Logical Exclusive OR Repeat String Operation Prefix F3 6C REP INS <i>m8</i> , DX Valid Valid Input (E)CX bytes from port DX into ES:[(E)DI].		
PMOVMSKB POP PSLLDQ PSRLDQ PUSH PUSHF PUSHFD PXOR REP REPE REPE REPZ	Compare packed doublewords in <i>mm/m64</i> and <i>mmx</i> for equality Compare packed doublewords in <i>xmm2/m128</i> and xmm1 for equality Move Byte Mask of <i>xmm or mmx</i> to <i>r32</i> . Pop a Value from the Stack Shift <i>xmm1</i> Double Quadword Left Logical by <i>imm8</i> bytes and by shifting in 0s. Shift Double Quadword Right Logical Push Word or Doubleword Onto the Stack Push EFLAGS Register onto the Stack Logical Exclusive OR Repeat String Operation Prefix F3 6C REP INS <i>m8</i> , DX Valid Valid F3 6C REP INS <i>m8</i> , DX Valid VAL. Input (E)CX bytes from port DX into ES:[(E)DI]. Input RCX bytes from port DX into [RDI].		
PMOVMSKB POP PSLLDQ PSRLDQ PUSH PUSHF PUSHFD PXOR REP REPE REPE REPE REPZ REPNE DEPNZ	Compare packed doublewords in <i>mm/m64</i> and <i>mmx</i> for equality Compare packed doublewords in <i>xmm2/m128</i> and xmm1 for equality Move Byte Mask of <i>xmm or mmx</i> to <i>r32</i> . Pop a Value from the Stack Shift <i>xmm1</i> Double Quadword Left Logical by <i>imm8</i> bytes and by shifting in 0s. Shift Double Quadword Right Logical Push Word or Doubleword Onto the Stack Push EFLAGS Register onto the Stack Logical Exclusive OR Repeat String Operation Prefix F3 6C REP INS <i>m8</i> , DX Valid Valid F3 6D REP INS <i>m8</i> , DX Valid N.E. F3 6D REP INS <i>m16</i> , DX Valid Valid Input (E)CX words from port DX into ES:[(E)DI]. Input (E)CX words from port DX into ES:[(E)DI].		
PMOVMSKB POP PSLLDQ PSRLDQ PUSHF PUSHFD PXOR REP REPE REPE REPZ REPNE REPNE REPNZ	Compare packed doublewords in <i>mm/m64</i> and <i>mmx</i> for equality Compare packed doublewords in <i>xmm2/m128</i> and xmm1 for equality Move Byte Mask of <i>xmm or mmx</i> to <i>r32</i> . Pop a Value from the Stack Shift <i>xmm1</i> Double Quadword Left Logical by <i>imm8</i> bytes and by shifting in 0s. Shift Double Quadword Right Logical Push Word or Doubleword Onto the Stack Push EFLAGS Register onto the Stack Logical Exclusive OR Repeat String Operation Prefix F3 6C REP INS <i>m8</i> , DX Valid Valid F3 6D REP INS <i>m16</i> , DX Valid Valid F3 6D REP INS <i>m32</i> , DX Valid Valid Input (E)CX bytes from port DX into ES:[(E)DI]. Input (E)CX words from port DX into ES:[(E)DI]. Input (E)CX doublewords from port DX into ES:[(E)DI].		
PMOVMSKB POP PSLLDQ PSRLDQ PUSHF PUSHFD PUSHFD PXOR REP REPE REPZ REPNE REPNE REPNZ	Compare packed doublewords in <i>mm/m64</i> and <i>mmx</i> for equality Compare packed doublewords in <i>xmm2/m128</i> and xmm1 for equality Move Byte Mask of <i>xmm or mmx</i> to <i>r32</i> . Pop a Value from the Stack Shift <i>xmm1</i> Double Quadword Left Logical by <i>imm8</i> bytes and by shifting in 0s. Shift Double Quadword Right Logical Push Word or Doubleword Onto the Stack Push EFLAGS Register onto the Stack Logical Exclusive OR Repeat String Operation Prefix F3 6C REP INS <i>m8</i> , DX Valid Valid F3 6D REP INS <i>m16</i> , DX Valid Valid F3 6D REP INS <i>m32</i> , DX Valid Valid F3 A4 REP MOVS <i>m8</i> , <i>m8</i> Valid Valid F3 6D REP INS <i>m8</i> , m8 Valid Valid F3 A4 REP MOVS <i>m8</i> , <i>m8</i> Valid Valid		
PMOVMSKB POP PSLLDQ PSRLDQ PUSHF PUSHFD PUSHFD PXOR REP REPE REPZ REPNE REPNE REPNE REPNZ	Compare packed doublewords in mm/m64 and mmx for equality Compare packed doublewords in xmm2/m128 and xmm1 for equality Move Byte Mask of xmm or mmx to r32. Pop a Value from the Stack Shift xmm1 Double Quadword Left Logical by imm8 bytes and by shifting in 0s. Shift Double Quadword Right Logical Push Word or Doubleword Onto the Stack Push EFLAGS Register onto the Stack Logical Exclusive OR Repeat String Operation Prefix F3 6C REP INS m8, DX Valid Valid Input (E)CX bytes from port DX into ES:[(E)DI]. F3 6D REP INS m8, DX Valid Valid Input (E)CX words from port DX into ES:[(E)DI]. F3 6D REP INS m32, DX Valid Valid Input (E)CX doublewords from port DX into ES:[(E)DI]. F3 A4 REP MOVS m8, m8 Valid Valid Move (E)CX words from DS:[(E)SI] to ES:[(E)DI]. F3 A5 REP MOVS m16, m16 Valid Valid Move (E)CX words from DS:[(E)SI] to ES:[(E)DI].		
PMOVMSKB POP PSLLDQ PSRLDQ PUSHF PUSHFD PUSHFD PXOR REP REPE REPZ REPNE REPNE REPNZ	Compare packed doublewords in <i>mm/m64</i> and <i>mmx</i> for equality Compare packed doublewords in <i>xmm2/m128</i> and xmm1 for equality Move Byte Mask of <i>xmm or mmx</i> to <i>r32</i> . Pop a Value from the Stack Shift Double Quadword Left Logical by <i>imm8</i> bytes and by shifting in 0s. Shift Double Quadword Right Logical Push Word or Doubleword Onto the Stack Push EFLAGS Register onto the Stack Logical Exclusive OR Repeat String Operation Prefix F3 6C REP INS <i>m8</i> , DX Valid Valid F3 6D REP INS <i>m16</i> , DX Valid Valid F3 6D REP INS <i>m32</i> , DX Valid Valid F3 A4 REP MOVS <i>m8</i> , <i>m8</i> Valid Valid F3 A5 REP MOVS <i>m16</i> , <i>m16</i> Valid Valid Move (E)CX words from DS:[(E)SI] to ES:[(E)DI]. F3 A5 REP MOVS <i>m32</i> , <i>m32</i> Valid Valid Move (E)CX doublewords fromDS:[(E)SI] to ES:[(E)DI].		
PMOVMSKB POP PSLLDQ PSRLDQ PUSHF PUSHFD PUSHFD PXOR REP REPE REPZ REPNE REPNE REPNZ	Compare packed doublewords in <i>mm/m64</i> and <i>mmx</i> for equality Compare packed doublewords in <i>xmm2/m128</i> and xmm1 for equality Move Byte Mask of <i>xmm or mmx</i> to <i>r32</i> . Pop a Value from the Stack Shift <i>xmm1</i> Double Quadword Left Logical by <i>imm8</i> bytes and by shifting in 0s. Shift Double Quadword Right Logical Push Word or Doubleword Onto the Stack Push EFLAGS Register onto the Stack Logical Exclusive OR Repeat String Operation Prefix F3 6C REP INS <i>m8</i> , DX Valid Valid Input (E)CX bytes from port DX into ES:[(E)DI]. F3 6D REP INS <i>m16</i> , DX Valid Valid Input (E)CX words from port DX into ES:[(E)DI.] F3 6D REP INS <i>m8</i> , <i>m8</i> Valid Valid Input (E)CX bytes from DX into ES:[(E)DI.] F3 6D REP INS <i>m8</i> , <i>m8</i> Valid Valid F3 6D REP INS <i>m8</i> , <i>m8</i> Valid Valid Valid F3 6D REP		
PMOVMSKB POP PSLLDQ PSRLDQ PUSHF PUSHFD PXOR REP REPE REPZ REPNE REPNE REPNZ	Compare packed doublewords in <i>mm/m64</i> and <i>mmx</i> for equality Compare packed doublewords in <i>xmm2/m128</i> and xmm1 for equality Move Byte Mask of <i>xmm or mmx</i> to <i>r32</i> . Pop a Value from the Stack Shift Double Quadword Left Logical by <i>imm8</i> bytes and by shifting in 0s. Shift Double Quadword Right Logical Push Word or Doubleword Onto the Stack Push EFLAGS Register onto the Stack Logical Exclusive OR Repeat String Operation Prefix F3 6C REP INS <i>m8</i> , DX Valid Valid Input (E)CX bytes from port DX into ES:[(E)DI]. F3 6D REP INS <i>m8</i> , DX Valid Valid Input (E)CX bytes from port DX into ES:[(E)DI]. F3 6D REP INS <i>m8</i> , DX Valid Valid Input (E)CX words from port DX into ES:[(E)DI]. F3 6D REP INS <i>m8</i> , DX Valid Valid Input (E)CX words from port DX into ES:[(E)DI]. F3 6D REP INS <i>m8</i> , DX Valid Valid Input (E)CX words from DS:[(E)SI] to ES:[(E)DI]. F3 A4 REP MOVS <i>m8</i> , <i>m8</i> Valid Valid Move (E)CX words from DS:[(E)SI] to ES:[(E)DI]. F3 A5 REP MOVS <i>m16</i> , <i>m16</i> Valid Valid Move (E)CX words from DS:[(E)SI] to ES:[(E)DI]. F3 6E REP OUTS DX, <i>r/m8</i> Valid Valid Output (E)CX words from DS:[(E)SI] to port DX. F3 6F REP OUTS DX, <i>r/m16</i> Valid Valid Output (E)CX words from DS:[(E)SI] to port DX. F3 6F REP OUTS DX, <i>r/m16</i> Valid Valid Output (E)CX words from DS:[(E)SI] to port DX. F3 6F REP OUTS DX, <i>r/m16</i> Valid Valid Output (E)CX words from DS:[(E)SI] to port DX.		
PMOVMSKB POP PSLLDQ PSRLDQ PUSHF PUSHF PUSHFD PXOR REP REPE REPZ REPNE REPNE REPNZ	Compare packed doublewords in mm/m64 and mmx for equality Compare packed doublewords in xmm2/m128 and xmm1 for equality Move Byte Mask of xmm or mmx to r32. Pop a Value from the Stack Shift xmm1 Double Quadword Left Logical by imm8 bytes and by shifting in 0s. Shift Double Quadword Right Logical Push Word or Doubleword Onto the Stack Push EFLAGS Register onto the Stack Logical Exclusive OR Repeat String Operation Prefix F3 6C REP INS m8, DX Valid Valid Input (E)CX bytes from port DX into ES:[(E)DI]. F3 6C REP INS m8, DX Valid Valid Input (E)CX words from port DX into [RDI]. F3 6D REP INS m3, DX Valid Valid Input (E)CX doublewords from port DX into ES:[(E)DI.] F3 6D REP INS m32, DX Valid Valid Input (E)CX doublewords from port DX into ES:[(E)DI.] F3 4A REP MOVS m8, m8 Valid Valid Move (E)CX bytes from DS:[(E)SI] to ES:[(E)DI]. F3 45 REP MOVS m32, m32 Valid Valid Move (E)CX words from DS:[(E)SI] to ES:[(E)DI]. F3 45 REP MOVS m32, m32 Valid Valid Move (E)CX doublewords fromDS:[(E)SI] to ES:[(E)DI]. F3 65 REP MOVS m32, m32 Valid Valid Output (E)CX words from DS:[(E)SI] to port DX. F3 65 REP MOVS m32, m32 Valid Valid Output (E)CX words from DS:[(E)SI] to port DX. F3 66 REP OUTS DX, r/m8 Valid Valid Output (E)CX words from DS:[(E)SI] to port DX.		
PMOVMSKB POP PSLLDQ PSRLDQ PUSHF PUSHF PUSHFD PXOR REP REPE REPZ REPNE REPNE REPNZ	Compare packed doublewords in mm/m64 and mmx for equality Compare packed doublewords in xmm2/m128 and xmm1 for equality Move Byte Mask of xmm or mmx to r32. Pop a Value from the Stack Shift xmm1 Double Quadword Left Logical by imm8 bytes and by shifting in 0s. Shift Double Quadword Right Logical Push Word or Doubleword Onto the Stack Push Word or Doubleword Onto the Stack Push EFLAGS Register onto the Stack Logical Exclusive OR Repeat String Operation Prefix F3 6C REP INS m8, DX Valid Valid Input (E)CX bytes from port DX into ES:[(E)DI]. F3 6C REP INS m8, DX Valid Valid Input (E)CX bytes from port DX into ES:[(E)DI]. F3 6D REP INS m8, DX Valid Valid Input (E)CX words from port DX into ES:[(E)DI]. F3 6D REP INS m32, DX Valid Valid Input (E)CX words from port DX into ES:[(E)DI]. F3 A5 REP MOVS m8, m8 Valid Valid Move (E)CX words from DS:[(E)SI] to ES:[(E)DI]. F3 A5 REP MOVS m32, m32 Valid Valid Move (E)CX doublewords from DS:[(E)SI] to ES:[(E)DI]. F3 6E REP OUTS DX, r/m8 Valid Valid Output (E)CX words from DS:[(E)SI] to port DX. F3 6F REP OUTS DX, r/m8 Valid Valid Output (E)CX words from DS:[(E)SI] to port DX. F3 6F REP OUTS DX, r/m8 Valid Valid Output (E)CX words from DS:[(E)SI] to port DX. F3 6F		
PMOVMSKB POP PSLLDQ PSRLDQ PUSHF PUSHFD PXOR REP REPE REPZ REPNE REPNE REPNZ	Compare packed doublewords in mm/m64 and mmx for equality Compare packed doublewords in xmm2/m128 and xmm1 for equality Move Byte Mask of xmm or mmx to r32. Pop a Value from the Stack Shift xmm1 Double Quadword Right Logical Push Word or Doubleword Onto the Stack Push EFLAGS Register onto the Stack Push EFLAGS Register onto the Stack Logical Exclusive OR Repeat String Operation Prefix F3 6C REP INS m8, DX Valid Valid Input (E)CX bytes from port DX into ES:[(E)DI]. F3 6D REP INS m8, DX Valid Valid Input (E)CX words from port DX into ES:[(E)DI]. F3 6D REP INS m32, DX Valid Valid Input (E)CX words from port DX into ES:[(E)DI]. F3 6A REP MOVS m8, m8 Valid Valid Input (E)CX words from DS:[(E)SI] to ES:[(E)DI]. F3 A4 REP MOVS m32, m32 Valid Valid Move (E)CX words from DS:[(E)SI] to ES:[(E)DI]. F3 A5 REP MOVS m16, m16 Valid Valid Move (E)CX words from DS:[(E)SI] to ES:[(E)DI]. F3 A5 REP OUTS DX, r/m8 Valid Valid Output (E)CX words from DS:[(E)SI] to port DX. F3 6F REP OUTS DX, r/m8 Valid Valid Unput (E)CX words from DS:[(E)SI] to port DX. F3 6F REP OUTS DX, r/m6 Valid Valid Unput (E)CX words from DS:[(E)SI] to port DX. F3 6F REP OUTS DX, r/m16 Valid Valid Unput (E)CX words from DS:[(E)SI] to AL. F3 AD R		
PMOVMSKB POP PSLLDQ PSRLDQ PUSHF PUSHFD PXOR REP REPE REPZ REPNE REPNE REPNZ	Compare packed doublewords in <i>mm/m64</i> and <i>mmx</i> for equality Compare packed doublewords in <i>xmm2/m128</i> and xmm1 for equality Move Byte Mask of <i>xmm or mmx</i> to <i>r32</i> . Pop a Value from the Stack Shift Double Quadword Right Logical by <i>imm8</i> bytes and by shifting in 0s. Shift Double Quadword Right Logical Push Word or Doubleword Onto the Stack Push EFLAGS Register onto the Stack Logical Exclusive OR Repeat String Operation Prefix F3 6C REP INS <i>m8</i> , DX Valid Valid Input (E)CX bytes from port DX into ES:[(E)DI]. F3 6D REP INS <i>m8</i> , DX Valid Valid Input (E)CX words from port DX into ES:[(E)DI]. F3 6D REP INS <i>m32</i> , DX Valid Valid Input (E)CX words from port DX into ES:[(E)DI]. F3 A4 REP MOVS <i>m8</i> , <i>m8</i> Valid Valid Move (E)CX bytes from DS:[(E)SI] to ES:[(E)DI]. F3 6F REP OUTS <i>m8</i> , <i>m8</i> Valid Valid Move (E)CX words from DS:[(E)SI] to ES:[(E)DI]. F3 6F REP OUTS DX, <i>r/m8</i> Valid Valid Output (E)CX words from DS:[(E)SI] to ES:[(E)DI]. F3 AC REP LODS AX Valid Valid Load (E)CX words from DS:[(E)SI] to AL. F3 AD REP LODS EAX Valid Valid Load (E)CX words from DS:[(E)SI] to AX. F3 AD REP LODS EAX Valid Valid Return from Procedure		
PMOVMSKB POP PSLLDQ PSRLDQ PUSH PUSHFD PXOR REP REPE REPZ REPNE REPNE REPNZ REPNZ	Compare packed doublewords in mm/m64 and mmx for equality Move Byte Mask of xmm or mmx to r32. Pop a Value from the Stack Shift xmm/ Double Quadword Left Logical by imm8 bytes and by shifting in 0s. Shift Xmm/ Double Quadword Right Logical Push Word or Doubleword Onto the Stack Push Word or Doubleword Onto the Stack Push EFLAGS Register onto the Stack Logical Exclusive OR Repeat String Operation Prefix F3 6C REP INS m8, DX Valid Valid Input (E)CX bytes from port DX into ES:[(E)DI]. F3 6C REP INS m8, DX Valid Valid Input RCX bytes from port DX into ES:[(E)DI]. F3 6D REP INS m8, DX Valid Valid Input (E)CX words from port DX into ES:[(E)DI]. F3 6D REP INS m8, DX Valid Valid Input (E)CX doublewords from port DX into ES:[(E)DI]. F3 6D REP INS m8, DX Valid Valid Input (E)CX words from DS:[[(E)SI] to ES:[(E)DI]. F3 6D REP INS m32, DX Valid Valid Move (E)CX words from DS:[[(E)SI] to ES:[(E)DI]. F3 A5 REP MOVS m8, m8 Valid Valid Move (E)CX doublewords from DS:[[(E)SI] to ES:[(E)DI]. F3 6E REP OUTS DX, r/m8 Valid Valid Output (E)CX words from DS:[[(E)SI] to port DX. F3 6F REP OUTS DX, r/m8 Valid Valid Output (E)CX words from DS:[[(E)SI] to port DX. F3 6F REP OUTS DX, r/m8 Valid Valid Load (E)CX words from DS:[[(E)SI] to AL.		
PMOVMSKB POP PSLLDQ PSRLDQ PUSHF PUSHFD PUSHFD PXOR REP REPE REPZ REPNE REPNE REPNZ REPNZ	Compare packed doublewords in <i>mm/m64</i> and <i>mmx</i> for equality Compare packed doublewords in <i>mm/m64</i> and <i>mmx</i> for equality Move Byte Mask of <i>xmm or mmx</i> to <i>r32</i> . Pop a Value from the Stack Shift <i>xmm1</i> Double Quadword Left Logical by <i>imm8</i> bytes and by shifting in 0s. Shift Double Quadword Right Logical Push Word or Doubleword Onto the Stack Push EFLAGS Register onto the Stack Logical Exclusive OR Repeat String Operation Prefix F3 6C REP INS <i>m8</i> , DX Valid Valid Input (E)CX bytes from port DX into ES:[(E)DI]. F3 6C REP INS <i>m8</i> , DX Valid Valid Input (E)CX words from port DX into ES:[(E)DI.] F3 6D REP INS <i>m16</i> , DX Valid Valid Input (E)CX words from Dort DX into ES:[(E)DI]. F3 6D REP INS <i>m3</i> , DX Valid Valid Input (E)CX words from Dort DX into ES:[(E)DI]. F3 6B REP MOVS <i>m8</i> , <i>m8</i> Valid Valid Move (E)CX words from DS:[(E)SI] to ES:[(E)DI]. F3 65 REP MOVS <i>m16</i> , <i>m16</i> Valid Valid Move (E)CX words from DS:[(E)SI] to ES:[(E)DI]. F3 65 REP MOVS <i>m32</i> , <i>m32</i> Valid Valid Output (E)CX words from DS:[(E)SI] to ES:[(E)DI]. F3 65 REP OUTS DX, <i>r/m16</i> Valid Valid Output (E)CX words from DS:[(E)SI] to port DX. F3 6F REP OUTS DX, <i>r/m16</i> Valid Valid Output (E)CX words from DS:[(E)SI] to port DX. F3 6F REP OUTS DX, <i>r/m16</i> Valid Valid Load (E)CX words from DS:[(E)SI] to port DX. F3 AC REP LODS AL Valid Valid Load (E)CX words from DS:[(E)SI] to AL. F3 AD REP LODS AX Valid Valid Load (E)CX words from DS:[(E)SI] to AL. F3 AD REP LODS AX Valid Valid Load (E)CX words from DS:[(E)SI] to AX. F3 AD REP LODS EAX Valid Valid Load (E)CX words from DS:[(E)SI] to AX. F3 AD REP LODS EAX Valid Valid Load (E)CX words from DS:[(E)SI] to AX. F3 AD REP LODS EAX Valid Valid Load (E)CX words from DS:[(E)SI] to AX. F3 AD REP LODS EAX Valid Valid Load (E)CX words from DS:[(E)SI] to AX. F3 AD REP LODS EAX Valid Valid Load (E)CX words from DS:[(E)SI] to AX. F3 AD REP LODS EAX Valid Valid Load (E)CX words from DS:[(E)SI] to AX. F3 AD REP LODS EAX Valid Valid Coad (E)CX words from DS:[(E)SI] to AX. F3 A		
PMOVMSKB POP PSLLDQ PSRLDQ PUSHF PUSHFD PUSHFD PXOR REP REPE REPZ REPNE REPNE REPNZ	Compare packed doublewords in xmm2/m128 and xmm1 for equality Move Byte Mask of xmm or mmx to r32. Pop a Value from the Stack Shift xmm1 Double Quadword Left Logical by imm8 bytes and by shifting in 0s. Shift xmm1 Double Quadword Right Logical Push Word or Doubleword Onto the Stack Push EFLAGS Register onto the Stack Logical Exclusive OR Repeat String Operation Prefix F3 6C REP INS m8, DX Valid Valid Input (E)CX bytes from port DX into ES:[(E)DI]. F3 6C REP INS m8, DX Valid Valid Input (E)CX words from port DX into ES:[(E)DI] F3 6D REP INS m8, DX Valid Valid Input (E)CX words from port DX into ES:[(E)DI]. F3 6D REP INS m3, DX Valid Valid Input (E)CX words from Dot DX into ES:[(E)DI]. F3 6A REP MOVS m8, m8 Valid Valid Move (E)CX bytes from DS:[(E)SI] to ES:[(E)DI]. F3 A4 REP MOVS m3, m8 Valid Valid Move (E)CX words from DS:[(E)SI] to ES:[(E)DI]. F3 A5 REP MOVS m16, m16 Valid Valid Output (E)CX words from DS:[(E)SI] to ES:[(E)DI]. F3 A5 REP MOVS m32, m32 Valid Valid Output (E)CX words from DS:[(E)SI] to port DX. F3 6F REP OUTS DX, r/m8 Valid Valid Load (E)CX words from DS:[(E)SI] to port DX. F3 A6 REP LODS AL Valid Valid Load (E)CX words from DS:[(E)SI] to AL. F3 A0 REP LODS AX Valid Valid Load (E)CX words from DS:[(E)SI] to AX.		
PMOVMSKB POP PSLLDQ PSRLDQ PUSHF PUSHFD PXOR REP REPE REPZ REPNE REPNE REPNZ	Compare packed doublewords in <i>xmm2/m128</i> and <i>xmm1</i> for equality Move Byte Mask of <i>xmm or mmx</i> to <i>r32</i> . Pop a Value from the Stack Shift <i>xmm1</i> Double Quadword Left Logical by <i>imm8</i> bytes and by shifting in 0s. Shift <i>num1</i> Double Quadword Right Logical Push Word or Doubleword Onto the Stack Push Word or Doubleword Onto the Stack Logical Exclusive OR Repeat String Operation Prefix F3 6C REP INS <i>m8</i> , DX Valid Valid Input (E)CX bytes from port DX into ES:[(E)DI]. F3 6D REP INS <i>m8</i> , DX Valid Valid Input (E)CX words from port DX into ES:[(E)DI]. F3 6D REP INS <i>m3</i> , DX Valid Valid Input (E)CX doublewords from port DX into ES:[(E)DI]. F3 6D REP INS <i>m3</i> , DX Valid Valid Input (E)CX words from port DX into ES:[(E)DI]. F3 4A REP MOVS <i>m3</i> , <i>m8</i> Valid Valid Move (E)CX words from DS:[(E)SI] to ES:[(E)DI]. F3 A5 REP MOVS <i>m3</i> , <i>m3</i> Valid Valid Move (E)CX words from DS:[(E)SI] to ES:[(E)DI]. F3 46 REP OUTS DX, <i>r/m8</i> Valid Valid Output (E)CX words from DS:[(E)SI] to ES:[(E)DI]. F3 66 REP OUTS DX, <i>r/m8</i> Valid Valid Load (E)CX words from DS:[(E)SI] to port DX. F3 66 REP OUTS DX, <i>r/m8</i> Valid Valid Output (E)CX words from DS:[(E)SI] to port DX. F3 66 REP OUTS DX, <i>r/m8</i> Valid Valid Load (E)CX words from DS:[(E)SI] to port DX. F3 66 REP OUTS DX, <i>r/m8</i> V		
PMOVMSKB POP PSLLDQ PSRLDQ PUSHF PUSHFD PXOR REP REPE REPZ REPNE REPNE REPNZ	Compare packed doublewords in <i>mm/m64</i> and <i>mmx</i> for equality Compare packed doublewords in <i>xmm2/m128</i> and xmm1 for equality Move Byte Mask of <i>xmm or mmx</i> to <i>r32</i> . Pop a Value from the Stack Shift <i>xmm1</i> Double Quadword Left Logical by <i>imm8</i> bytes and by shifting in 0s. Shift Double Quadword Right Logical Push Word or Doubleword Onto the Stack Push EFLAGS Register onto the Stack Logical Exclusive OR Repeat String Operation Prefix F3 6C REP INS <i>m8</i> , DX Valid Valid Input (E)CX bytes from port DX into ES:[(E)DI]. F3 6C REP INS <i>m8</i> , DX Valid Valid Input RCX bytes from port DX into ES:[(E)DI]. F3 6D REP INS <i>m16</i> , DX Valid Valid Input (E)CX words from port DX into ES:[(E)DI]. F3 45 REP MOVS <i>m32</i> , <i>m32</i> Valid Valid Move (E)CX words from DS:[(E)SI] to ES:[(E)DI]. F3 45 REP MOVS <i>m32</i> , <i>m32</i> Valid Valid Move (E)CX words from DS:[(E)SI] to ES:[(E)DI]. F3 65 REP OUTS DX, <i>r/m8</i> Valid Valid Output (E)CX words from DS:[(E)SI] to ES:[(E)DI]. F3 66 REP LODS AL Valid Valid Load (E)CX words from DS:[(E)SI] to EX. F3 67 REP OUTS DX, <i>r/m16</i> Valid Valid Load (E)CX words from DS:[(E)SI] to AL. F3 40 REP LODS AX Valid Valid Load (E)CX words from DS:[(E)SI] to AL. F3 40 REP LODS AX Valid Valid Load (E)		
PMOVMSKB POP PSLLDQ PSRLDQ PUSHF PUSHFD PXOR REP REPE REPZ REPNE REPNE REPNZ	Compare packed doublewords in <i>xmm/m64</i> and <i>mmx</i> for equality Move Byte Mask of <i>xmm or mmx</i> to <i>r32</i> . Pop a Value from the Stack Shift <i>xmm1</i> Double Quadword Left Logical by <i>imm8</i> bytes and by shifting in 0s. Shift <i>xmm1</i> Double Quadword Right Logical Push Word or Doubleword Onto the Stack Push FLAGS Register onto the Stack Logical Exclusive OR Repeat String Operation Prefix F3 6C REP INS <i>m8</i> , DX Valid Valid Input (E)CX bytes from port DX into ES:[(E)DI]. F3 6D REP INS <i>m8</i> , DX Valid Valid Input (E)CX words from port DX into ES:[(E)DI]. F3 6D REP INS <i>m32</i> , DX Valid Valid Input (E)CX words from port DX into ES:[(E)DI]. F3 4A REP MOVS <i>m8</i> , <i>m8</i> Valid Valid Input (E)CX words from DS:[(E)SI] to ES:[(E)DI]. F3 45 REP MOVS <i>m8</i> , <i>m8</i> Valid Valid Move (E)CX doublewords from DS:[(E)SI] to ES:[(E)DI]. F3 65 REP MOVS <i>m3</i> , <i>m32</i> Valid Valid Move (E)CX words from DS:[(E)SI] to port DX. F3 66 REP OUTS DX, <i>r/m8</i> Valid Valid Output (E)CX words from DS:[(E)SI] to port DX. F3 66 REP MOVS <i>m3</i> , <i>m32</i> Valid Valid Output (E)CX words from DS:[(E)SI] to port DX. F3 67 REP OUTS DX, <i>r/m8</i> Valid Valid Load (E)CX bytes from DS:[(E)SI] to port DX. F3 68 REP MOVS AX valid Valid Load (E)CX words from DS:[(E)SI] to port DX. F3 60 REP LODS AX Valid Valid Load (E)CX		

	Shift <i>mill</i> to right CL places while shifting bits from <i>r16</i> in from the left.			
	Shift <i>r/m32</i> to right <i>imm8</i> places while shifting bits from <i>r32</i> in from the left.			
	Shift <i>r/m32</i> to right CL places while shifting bits from <i>r32</i> in from the left.			
SHUFPD	Shuffle Packed Double-precision Floating-Point Values			
	Shuffle packed double-precision floating-point values selected by <i>imm8</i>			
	from <i>xmm1</i> and <i>xmm2/m128</i> to <i>xmm1</i> .			
	DEST X1 X0			
	SRC 11			
	DEST Y1 or YD X1 or X0			
	The source operand can be an XMM register or a 128-bit memory location.			
	The destination operand is an XMM register. The select operand is an 8-bit immediate: bit 0 selects which			
	value is moved from the destination operand to the result (where 0 selects the low quadword and 1 selects the			
	high quadword) and bit 1 selects which value is moved from the source operand to the result. Bits 2 through 7			
	of the select operand are reserved and must be set to 0.			
SQRTSD	Compute Square Root of Scalar Double-precision Floating-Point Value			
-	Computes square root of the low double-precision floatingpoint value in <i>xmm2/m64</i> and			
	stores the results in <i>xmm1</i> .			
STOS	Store String			
STOSB	store AL at address ES:(E)DI;			
STOSW	store AX at address ES:(E)DI;			
STOSD	store AL at address ES:(E)DI:			
	store AX at address ES:(E)DI:			
	store EAX at address ES:(E)DI;			
SUBPD	Subtract Packed Double-precision Floating-Point Values			
	in <i>xmm2/m128</i> from <i>xmm1</i>			
SUBSD	Subtract Scalar Double-precision Floating-Point Values			
	in <i>xmm21/m64</i> from <i>xmm1</i>			
	Subtracts the low double-precision floating-point values in $xmm^2/mem64$ from xmm^1 .			
UCOMISD	Unordered Compare Scolar Double precision Electing Doint Voluce part 1 and and 24.64 and			
	Unordered Compare Scalar Double-precision Floating-Point Values II xmm/ and xmm2/m04 and			
	Set EFLAGS			
	Set EFLAGS Performs and unordered compare of the double-precision floating-point values in the low quadwords of source			
	Set EFLAGS Performs and unordered compare of the double-precision floating-point values in the low quadwords of source operand 1 (first operand) and source operand 2 (second operand), and sets the ZF, PF, and CF flags in the			
	Set EFLAGS Performs and unordered compare of the double-precision floating-point values in the low quadwords of source operand 1 (first operand) and source operand 2 (second operand), and sets the ZF, PF, and CF flags in the EFLAGS register according to the result (unordered, greater than, less than, or equal). The OF, SF and AF flags			
	Set EFLAGS Performs and unordered compare of the double-precision floating-point values in the low quadwords of source operand 1 (first operand) and source operand 2 (second operand), and sets the ZF, PF, and CF flags in the EFLAGS register according to the result (unordered, greater than, less than, or equal). The OF, SF and AF flags in the EFLAGS register are set to 0. The unordered result is returned if either source operand is a NaN (QNaN			
	Set EFLAGS Performs and unordered compare of the double-precision floating-point values in the low quadwords of source operand 1 (first operand) and source operand 2 (second operand), and sets the ZF, PF, and CF flags in the EFLAGS register according to the result (unordered, greater than, less than, or equal). The OF, SF and AF flags in the EFLAGS register are set to 0. The unordered result is returned if either source operand is a NaN (QNaN or SNaN). Source operand 1 is an XMM register; source operand 2 can be an XMM register or a 64 bit memory			
	Set EFLAGS Performs and unordered compare of the double-precision floating-point values in the low quadwords of source operand 1 (first operand) and source operand 2 (second operand), and sets the ZF, PF, and CF flags in the EFLAGS register according to the result (unordered, greater than, less than, or equal). The OF, SF and AF flags in the EFLAGS register are set to 0. The unordered result is returned if either source operand is a NaN (QNaN or SNaN). Source operand 1 is an XMM register; source operand 2 can be an XMM register or a 64 bit memory location.			
UNPCKHPD	Set EFLAGS Performs and unordered compare of the double-precision floating-point values in the low quadwords of source operand 1 (first operand) and source operand 2 (second operand), and sets the ZF, PF, and CF flags in the EFLAGS register according to the result (unordered, greater than, less than, or equal). The OF, SF and AF flags in the EFLAGS register are set to 0. The unordered result is returned if either source operand is a NaN (QNaN or SNaN). Source operand 1 is an XMM register; source operand 2 can be an XMM register or a 64 bit memory location.			
UNPCKHPD	 Set EFLAGS Performs and unordered compare of the double-precision floating-point values in the low quadwords of source operand 1 (first operand) and source operand 2 (second operand), and sets the ZF, PF, and CF flags in the EFLAGS register according to the result (unordered, greater than, less than, or equal). The OF, SF and AF flags in the EFLAGS register are set to 0. The unordered result is returned if either source operand is a NaN (QNaN or SNaN). Source operand 1 is an XMM register; source operand 2 can be an XMM register or a 64 bit memory location. Unpack and Interleave High Packed Double-precision Floating-Point Values UNPCKHPD xmm1, xmm2/m128 			
UNPCKHPD	Set EFLAGS Performs and unordered compare of the double-precision floating-point values in the low quadwords of source operand 1 (first operand) and source operand 2 (second operand), and sets the ZF, PF, and CF flags in the EFLAGS register according to the result (unordered, greater than, less than, or equal). The OF, SF and AF flags in the EFLAGS register are set to 0. The unordered result is returned if either source operand is a NaN (QNaN or SNaN). Source operand 1 is an XMM register; source operand 2 can be an XMM register or a 64 bit memory location. Unpack and Interleave High Packed Double-precision Floating-Point Values UNPCKHPD xmm1, xmm2/m128			
UNPCKHPD	Set EFLAGS Performs and unordered compare of the double-precision floating-point values in the low quadwords of source operand 1 (first operand) and source operand 2 (second operand), and sets the ZF, PF, and CF flags in the EFLAGS register according to the result (unordered, greater than, less than, or equal). The OF, SF and AF flags in the EFLAGS register are set to 0. The unordered result is returned if either source operand is a NaN (QNaN or SNaN). Source operand 1 is an XMM register; source operand 2 can be an XMM register or a 64 bit memory location. Unpack and Interleave High Packed Double-precision Floating-Point Values UNPCKHPD xmm1, xmm2/m128 DEst x1			
UNPCKHPD	Set EFLAGS Performs and unordered compare of the double-precision floating-point values in the low quadwords of source operand 1 (first operand) and source operand 2 (second operand), and sets the ZF, PF, and CF flags in the EFLAGS register according to the result (unordered, greater than, less than, or equal). The OF, SF and AF flags in the EFLAGS register are set to 0. The unordered result is returned if either source operand is a NaN (QNaN or SNaN). Source operand 1 is an XMM register; source operand 2 can be an XMM register or a 64 bit memory location. Unpack and Interleave High Packed Double-precision Floating-Point Values UNPCKHPD xmm1, xmm2/m128 DEST x1 X1 X0			
UNPCKHPD	Set EFLAGS Performs and unordered compare of the double-precision floating-point values in the low quadwords of source operand 1 (first operand) and source operand 2 (second operand), and sets the ZF, PF, and CF flags in the EFLAGS register according to the result (unordered, greater than, less than, or equal). The OF, SF and AF flags in the EFLAGS register are set to 0. The unordered result is returned if either source operand is a NaN (QNaN or SNaN). Source operand 1 is an XMM register; source operand 2 can be an XMM register or a 64 bit memory location. Unpack and Interleave High Packed Double-precision Floating-Point Values UNPCKHPD xmm1, xmm2/m128 DEST X1 SRC Y1 Y0			
UNPCKHPD	Set EFLAGS Performs and unordered compare of the double-precision floating-point values in the low quadwords of source operand 1 (first operand) and source operand 2 (second operand), and sets the ZF, PF, and CF flags in the EFLAGS register according to the result (unordered, greater than, less than, or equal). The OF, SF and AF flags in the EFLAGS register are set to 0. The unordered result is returned if either source operand is a NaN (QNaN or SNaN). Source operand 1 is an XMM register; source operand 2 can be an XMM register or a 64 bit memory location. Unpack and Interleave High Packed Double-precision Floating-Point Values UNPCKHPD xmm1, xmm2/m128 DEST x1 X1 X0			
UNPCKHPD	Set EFLAGS Performs and unordered compare of the double-precision floating-point values in the low quadwords of source operand 1 (first operand) and source operand 2 (second operand), and sets the ZF, PF, and CF flags in the EFLAGS register according to the result (unordered, greater than, less than, or equal). The OF, SF and AF flags in the EFLAGS register are set to 0. The unordered result is returned if either source operand is a NaN (QNaN or SNaN). Source operand 1 is an XMM register; source operand 2 can be an XMM register or a 64 bit memory location. Unpack and Interleave High Packed Double-precision Floating-Point Values UNPCKHPD xmm1, xmm2/m128 DEST X1 X1 X0			
UNPCKHPD	Ontotale of compare Scalar Double-precision Floating-Point Values II Xmm1 and Xmm2/m04 and Set EFLAGSPerforms and unordered compare of the double-precision floating-point values in the low quadwords of source operand 1 (first operand) and source operand 2 (second operand), and sets the ZF, PF, and CF flags in the EFLAGS register according to the result (unordered, greater than, less than, or equal). The OF, SF and AF flags in the EFLAGS register are set to 0. The unordered result is returned if either source operand is a NaN (QNaN or SNaN). Source operand 1 is an XMM register; source operand 2 can be an XMM register or a 64 bit memory location.Unpack and Interleave High Packed Double-precision Floating-Point Values UNPCKHPD xmm1, xmm2/m128DESTx1x1x0V1V0V1X1			
UNPCKHPD	Set EFLAGS Performs and unordered compare of the double-precision floating-point values in the low quadwords of source operand 1 (first operand) and source operand 2 (second operand), and sets the ZF, PF, and CF flags in the EFLAGS register according to the result (unordered, greater than, less than, or equal). The OF, SF and AF flags in the EFLAGS register are set to 0. The unordered result is returned if either source operand is a NaN (QNaN or SNaN). Source operand 1 is an XMM register; source operand 2 can be an XMM register or a 64 bit memory location. Unpack and Interleave High Packed Double-precision Floating-Point Values UNPCKHPD xmm1, xmm2/m128 DEST X1 Y1 Y0 Unpack and Interleave Low Packed Double-precision Floating-Point Values Unpack and Interleave Low Packed Double-precision Floating-Point Values			
UNPCKHPD	Ontordered Compare Scalar Double-precision Floating-Point Values II Xmm1 and Xmm2/m04 and Set EFLAGS Performs and unordered compare of the double-precision floating-point values in the low quadwords of source operand 1 (first operand) and source operand 2 (second operand), and sets the ZF, PF, and CF flags in the EFLAGS register according to the result (unordered, greater than, less than, or equal). The OF, SF and AF flags in the EFLAGS register are set to 0. The unordered result is returned if either source operand is a NaN (QNaN or SNaN). Source operand 1 is an XMM register; source operand 2 can be an XMM register or a 64 bit memory location. Unpack and Interleave High Packed Double-precision Floating-Point Values UNPCKHPD xmm1, xmm2/m128 DEST XI YI YI <td c<="" th=""></td>			
UNPCKHPD	Output Scalar Double-precision Floating-Point values if xmm1 and xmm2/m04 and Set EFLAGS Performs and unordered compare of the double-precision floating-point values in the low quadwords of source operand 1 (first operand) and source operand 2 (second operand), and sets the ZF, PF, and CF flags in the EFLAGS register according to the result (unordered, greater than, less than, or equal). The OF, SF and AF flags in the EFLAGS register are set to 0. The unordered result is returned if either source operand is a NaN (QNaN or SNaN). Source operand 1 is an XMM register; source operand 2 can be an XMM register or a 64 bit memory location. Unpack and Interleave High Packed Double-precision Floating-Point Values UNPCKHPD xmm1, xmm2/m128 DEST XI XI XI VI VI VI VII <			
UNPCKHPD	Onordered compare Scalar Double-precision Floating-Point values in xmm1 and xmm2/m04 and Set EFLAGS Performs and unordered compare of the double-precision floating-point values in the low quadwords of source operand 1 (first operand) and source operand 2 (second operand), and sets the ZF, PF, and CF flags in the EFLAGS register according to the result (unordered, greater than, less than, or equal). The OF, SF and AF flags in the EFLAGS register are set to 0. The unordered result is returned if either source operand is a NaN (QNaN or SNaN). Source operand 1 is an XMM register; source operand 2 can be an XMM register or a 64 bit memory location. Unpack and Interleave High Packed Double-precision Floating-Point Values UNPCKHPD xmm1, xmm2/m128 DEST XI VI VI <td co<="" th=""></td>			
UNPCKHPD	Onordered compare Scalar Double-precision Floating-Point values in xmm1 and xmm2/m04 and Set EFLAGS Performs and unordered compare of the double-precision floating-point values in the low quadwords of source operand 1 (first operand) and source operand 2 (second operand), and sets the ZF, PF, and CF flags in the EFLAGS register according to the result (unordered, greater than, less than, or equal). The OF, SF and AF flags in the EFLAGS register are set to 0. The unordered result is returned if either source operand is a NaN (QNaN or SNaN). Source operand 1 is an XMM register; source operand 2 can be an XMM register or a 64 bit memory location. Unpack and Interleave High Packed Double-precision Floating-Point Values UNPCKHPD xmm1, xmm2/m128 DEST XI XI XI VI VI <td co<="" th=""></td>			
UNPCKHPD	Ontoteled compare Scalar Double-precision Profiling-Point Values II xmm1 and xmm2/m64 and Set EFLAGS Performs and unordered compare of the double-precision floating-point values in the low quadwords of source operand 1 (first operand) and source operand 2 (second operand), and sets the ZF, PF, and CF flags in the EFLAGS register according to the result (unordered, greater than, less than, or equal). The OF, SF and AF flags in the EFLAGS register are set to 0. The unordered result is returned if either source operand is a NaN (QNaN or SNaN). Source operand 1 is an XMM register; source operand 2 can be an XMM register or a 64 bit memory location. Unpack and Interleave High Packed Double-precision Floating-Point Values UNPCKHPD xmm1, xmm2/m128 DEST X1 VI VI <td c<="" th=""></td>			
UNPCKHPD	Ontoteled compare Scalar Double-precision Floating-Point Values II Xmm1 and Xmm2/m64 and Set EFLAGS Set EFLAGS Performs and unordered compare of the double-precision floating-point values in the low quadwords of source operand 1 (first operand) and source operand 2 (second operand), and sets the ZF, PF, and CF flags in the EFLAGS register are set to 0. The unordered result is returned if either source operand is a NaN (QNaN or SNaN). Source operand 1 is an XMM register; source operand 2 can be an XMM register or a 64 bit memory location. Unpack and Interleave High Packed Double-precision Floating-Point Values UNPCKHPD xmm1, xmm2/m128 DEST X1 X0 Dest Y1 Y			
UNPCKHPD	Set EFLAGS Performs and unordered compare of the double-precision floating-point values in the low quadwords of source operand 1 (first operand) and source operand 2 (second operand), and sets the ZF, PF, and CF flags in the EFLAGS register according to the result (unordered, greater than, less than, or equal). The OF, SF and AF flags in the EFLAGS register are set to 0. The unordered result is returned if either source operand is a NaN (QNaN or SNaN). Source operand 1 is an XMM register; source operand 2 can be an XMM register or a 64 bit memory location. UNPack and Interleave High Packed Double-precision Floating-Point Values UNPCKHPD xmm1, xmm2/m128 DEST x1 x0 SRC y1 y0 SRC y1 SRC y1 S			
UNPCKHPD UNPCKLPD	Ondered compare Scalar Double-precision Floating-Point values in xmm1 and xmm12m04 and set Set EFLAGS Set EFLAGS Performs and unordered compare of the double-precision floating-point values in the low quadwords of source operand 1 (first operand) and source operand 2 (second operand), and sets the ZF, PF, and CF flags in the EFLAGS register according to the result (unordered, greater than, less than, or equal). The OF, SF and AF flags in the EFLAGS register are set to 0. The unordered result is returned if either source operand is a NaN (QNaN or SNaN). Source operand 1 is an XMM register; source operand 2 can be an XMM register or a 64 bit memory location. Unpack and Interleave High Packed Double-precision Floating-Point Values UNPCKHPD xmm1, xmm2/m128 DEST XI XI XI XI VI			
UNPCKHPD UNPCKLPD VERR VERW	Ondered compare Scalar Double-precision Floating-Point Values in <i>Anim2</i> and <i>Anim2</i> /m04 and Set EFLAGS Set EFLAGS Performs and unordered compare of the double-precision floating-point values in the low quadwords of source operand 1 (first operand) and source operand 2 (second operand), and sets the ZF, PF, and CF flags in the EFLAGS register are set to 0. The unordered result is returned if either source operand is a NaN (QNaN or SNaN). Source operand 1 is an XMM register; source operand 2 can be an XMM register or a 64 bit memory location. Unpack and Interleave High Packed Double-precision Floating-Point Values UNPCKHPD xmm1, xmm2/m128 Uses N Set Colspan="2">Colspan="2"Colspan="2"Colspan="2"Colspan="2"Colspan="2"Colspan="2"Colspan="2			
UNPCKHPD UNPCKLPD VERR VERW	Ondered compare Scalar Double-precision Proating-Point Values II xmm1 and xmm2/mo4 and Set EFLAGS Performs and unordered compare of the double-precision floating-point values in the low quadwords of source operand 1 (first operand) and source operand 2 (second operand), and sets the ZF, PF, and CF flags in the EFLAGS register according to the result (unordered, greater than, less than, or equal). The OF, SF and AF flags in the EFLAGS register are set to 0. The unordered result is returned if either source operand is a NaN (QNaN or SNaN). Source operand 1 is an XMM register; source operand 2 can be an XMM register or a 64 bit memory location. Unpack and Interleave High Packed Double-precision Floating-Point Values UNPCKHPD xmm1, xmm2/m128 USEST x1 v1 x0 v2 x0 Verify a Segment for Reading Verify a Segment for Writing Verifies whether the code or data segment specified with the source operand is readable (VERR) or writable			
UNPCKHPD UNPCKLPD VERR VERW	Ondered compare Scalar Double-precision Floating-Point Values II xmm1 and xmm2/mo4 and Set EFLAGS Performs and unordered compare of the double-precision floating-point values in the low quadwords of source operand 1 (first operand) and source operand 2 (second operand), and sets the ZF, PF, and CF flags in the EFLAGS register are set to 0. The unordered result is returned if either source operand is a NaN (QNaN or SNaN). Source operand 1 is an XMM register; source operand 2 can be an XMM register or a 64 bit memory location. Unpack and Interleave High Packed Double-precision Floating-Point Values UNPCKHPD xmm1, xmm2/m128 UEST VI VI VI VI VI VI VI VIII VIIII VIII VIIIII VIIII VIIIIIIIIIIIIIIIIIIIIIIIIIIIIIII			

	PentDoublePrecFPCompact.doc
	(VERR) or writable (VERW), the ZF flag is set; otherwise, the ZF flag is cleared. Code segments are never
	verified as writable. This check cannot be performed on system segments.
WAIT =	Causes the processor to check for and handle pending, unmasked, floating-point exceptions before proceeding
FWAIT	
XORPD	Bitwise Logical XOR for Double-precision Floating-Point Values
	Performs a bitwise logical exclusive-OR of the two packed double-precision floating-point values from the
	source operand (second operand) and the destination operand (first operand), and stores the result in the
	destination operand. The source operand can be an XMM register or a 128-bit memory location. The
	destination operand is an XMM register.

JA	rel8	
JAE	rel8	
JB	rel8	
JBE	rel8	
JC	rel§	
JCXZ	relĝ	
JECXZ	relĝ	
JRCXZ	relĝ	
JE	relĝ	
JG	relĝ	
JGE	rel8	
JL	rel8	
JLE	rel8	
JNA	rel8	
JNAE	rel8	
JNAE JNB	relß relß	
JNAE JNB JNBE	relå relå relå	
JNAE JNB JNBE JNC	relß relß relß relß	
JNAE JNB JNBE JNC JNE	relå relå relå relå relå	
JNAE JNB JNBE JNC JNE JNG	relå relå relå relå relå relå	
JNAE JNB JNBE JNC JNC JNC JNG JNGE	relå relå relå relå relå relå relå	

JNLE	rel8	
JNO	rel§	-
JNP	rel8	-
JNS	rel8	-
JNZ	rel§	-
JO	rel§	-
JP	relĝ	-
JPE	rel§	-
JPO	rel8	-
JS	rel§	-
JZ	rel8	-
JA	relļģ	-
JA	rel32	-
JAE	rellØ	-
JAE	rel32	-
JB	rellØ	-
JB	rel32	-
JBE	relļģ	-
JBE	rel32	
JC	relļģ	
JC	rel32	
JE	rel]Ø	

JE	reigi
JZ	rellØ
JZ	rel32
JG	rellØ
JG	rel32
JGE	relļģ
JGE	rel32
JL	relļģ
JL	rel32
JLE	rellØ
JLE	rel32
JNA	relļģ
JNA	rel32
JNAE	rellø
JNAE	rel32
JNB	rellØ
JNB	rel32
JNBE	relļģ
JNBE	rel32
JNC	relļģ
JNC	rel32
JNE	rellØ

JNE	rel32	
JNG	relló	
JNG	rel32	
JNGE	relló	
JNGE	rel32	
JNL	rello	
JNL	rel32	
JNLE	relló	
JNLE	rel32	
JNO	relló	
JNO	rel32	
JNP	rello	
JNP	rel32	
JNS	relló	
JNS	rel32	
JNZ	relló	
JNZ	rel32	
JO	rello	
JO	rel32	
JP	relló	
JP	rel32	
JPE	relló	

JPE	rel32
JPO	rellØ
JPO	rel32
JS	rellØ
JS	rel32
JZ	rellØ
JZ	rel32
JMP	rel§
JMP	rellØ
JMP	rel32
JMP	r/ml0
JMP	r/m32
JMP	r/m04
JMP	ptr10:10
JMP	ptr10:32
JMP	m10:10
JMP	m16:32
JMP	m10:04