```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE oplist SYSTEM "oplist.dtd">
<!-- Automatically generated 2004-12-18 03:25:03 -0800 from ../reference/25366714.pdf -->

<oplist>
    <prefix bitmask="01100111" detail="address size" />
    <prefix bitmask="11110000" detail="LOCK" />
    <prefix bitmask="01100110" detail="operand size" />
    <prefix bitmask="00101110" detail="CS segment override" />
    <prefix bitmask="00111110" detail="DS segment override" />
    <prefix bitmask="00100110" detail="ES segment override" />
    <prefix bitmask="01100100" detail="FS segment override" />
    <prefix bitmask="01100101" detail="GS segment override" />
    <prefix bitmask="00110110" detail="SS segment override" />
    <op bitmask="00110111" mnemonic="AAA">
        <description>ASCII Adjust after Addition</description>
    </op>
    <op bitmask="1101010100001010" mnemonic="AAD">
        <description>ASCII Adjust AX before Division</description>
    </op>
    <op bitmask="1101010000001010" mnemonic="AAM">
        <description>ASCII Adjust AX after Multiply</description>
    </op>
    <op bitmask="00111111" mnemonic="AAS">
        <description>ASCII Adjust AL after Subtraction</description>
    </op>
    <op bitmask="0001000" mnemonic="ADC" detail="register to memory">
        <arg direction="input" type="reg" />
        <arg direction="output" type="mem" />
        <description>ADD with Carry</description>
    </op>
    <op bitmask="0001000x11" mnemonic="ADC" detail="register1 to register2">
        <arg direction="input" type="reg" />
        <arg direction="output" type="reg" />
        <description>ADD with Carry</description>
    </op>
    <op bitmask="0001001" mnemonic="ADC" detail="memory to register">
        <arg direction="input" type="mem" />
        <arg direction="output" type="reg" />
        <description>ADD with Carry</description>
    </op>
    <op bitmask="0001001x11" mnemonic="ADC" detail="register2 to register1">
        <arg direction="input" type="reg" />
        <arg direction="output" type="reg" />
        <description>ADD with Carry</description>
    </op>
    <op bitmask="0001010" mnemonic="ADC" detail="immediate to AL, AX, or EAX">
        <arg direction="input" type="imm" />
        <arg direction="output" type="reg" />
        <description>ADD with Carry</description>
    </op>
    <op bitmask="100000xx11010" mnemonic="ADC" detail="immediate to register">
        <arg direction="input" type="imm" />
        <arg direction="output" type="reg" />
        <description>ADD with Carry</description>
    </op>
    <op bitmask="100000xxxx010" mnemonic="ADC" detail="immediate to memory">
        <arg direction="input" type="imm" />
        <arg direction="output" type="mem" />
        <description>ADD with Carry</description>
    </op>
    <op bitmask="0000000" mnemonic="ADD" detail="register to memory">
        <arg direction="input" type="reg" />
        <arg direction="output" type="mem" />
        <description>Add</description>
```

```
    </op>
    <op bitmask="0000000x11" mnemonic="ADD" detail="register1 to register2">
        <arg direction="input" type="reg" />
        <arg direction="output" type="reg" />
        <description>Add</description>
    </op>
    <op bitmask="0000001" mnemonic="ADD" detail="memory to register">
        <arg direction="input" type="mem" />
        <arg direction="output" type="reg" />
        <description>Add</description>
    </op>
    <op bitmask="0000001x11" mnemonic="ADD" detail="register2 to register1">
        <arg direction="input" type="reg" />
        <arg direction="output" type="reg" />
        <description>Add</description>
    </op>
    <op bitmask="0000010" mnemonic="ADD" detail="immediate to AL, AX, or EAX">
        <arg direction="input" type="imm" />
        <arg direction="output" type="reg" />
        <description>Add</description>
    </op>
    <op bitmask="100000xx11000" mnemonic="ADD" detail="immediate to register">
        <arg direction="input" type="imm" />
        <arg direction="output" type="reg" />
        <description>Add</description>
    </op>
    <op bitmask="100000xxxx000" mnemonic="ADD" detail="immediate to memory">
        <arg direction="input" type="imm" />
        <arg direction="output" type="mem" />
        <description>Add</description>
    </op>
    <op bitmask="011001100000111101011000" mnemonic="ADDPD" detail="mem to xmmreg">
        <arg direction="input" type="mem" />
        <arg direction="output" type="xmmreg" />
        <description>Add Packed Double-Precision Floating-Point Values</description>
    </op>
    <op bitmask="011001100000111110101100011" mnemonic="ADDPD" detail="xmmreg to xmmreg">
        <arg direction="input" type="xmmreg" />
        <arg direction="output" type="xmmreg" />
        <description>Add Packed Double-Precision Floating-Point Values</description>
    </op>
    <op bitmask="0000111101011000" mnemonic="ADDPS" detail="mem to xmmreg">
        <arg direction="input" type="mem" />
        <arg direction="output" type="xmmreg" />
        <description>Add Packed Single-Precision Floating-Point Values</description>
    </op>
    <op bitmask="000011110101100011" mnemonic="ADDPS" detail="xmmreg to xmmreg">
        <arg direction="input" type="xmmreg" />
        <arg direction="output" type="xmmreg" />
        <description>Add Packed Single-Precision Floating-Point Values</description>
    </op>
    <op bitmask="111100100000111101011000" mnemonic="ADDSD" detail="mem to xmmreg">
        <arg direction="input" type="mem" />
        <arg direction="output" type="xmmreg" />
        <description>Add Scalar Double-Precision Floating-Point Values</description>
    </op>
    <op bitmask="111100100000111110101100011" mnemonic="ADDSD" detail="xmmreg to xmmreg">
        <arg direction="input" type="xmmreg" />
        <arg direction="output" type="xmmreg" />
        <description>Add Scalar Double-Precision Floating-Point Values</description>
    </op>
    <op bitmask="111100110000111101011000" mnemonic="ADDSS" detail="mem to xmmreg">
        <arg direction="input" type="mem" />
        <arg direction="output" type="xmmreg" />
        <description>Add Scalar Single-Precision Floating-Point Values</description>
```

```
    </op>
    <op bitmask="11110011000011110101100011" mnemonic="ADDSS" detail="xmmreg to xmmreg">
        <arg direction="input" type="xmmreg" />
        <arg direction="output" type="xmmreg" />
        <description>Add Scalar Single-Precision Floating-Point Values</description>
    </op>
    <op bitmask="01100110000011111010000" mnemonic="ADDSUBPD" detail="mem to xmmreg">
        <arg direction="input" type="mem" />
        <arg direction="output" type="xmmreg" />
        <description>Add /Sub packed DP FP numbers from XMM2/Mem to XMM1</description>
    </op>
    <op bitmask="01100110000011111101000011" mnemonic="ADDSUBPD" detail="xmmreg2 to xmmreg1"
>
        <arg direction="input" type="xmmreg" />
        <arg direction="output" type="xmmreg" />
        <description>Add /Sub packed DP FP numbers from XMM2/Mem to XMM1</description>
    </op>
    <op bitmask="11110010000011111010000" mnemonic="ADDSUBPS" detail="mem to xmmreg">
        <arg direction="input" type="mem" />
        <arg direction="output" type="xmmreg" />
        <description>Add /Sub packed SP FP numbers from XMM2/Mem to XMM1</description>
    </op>
    <op bitmask="11110010000011111101000011" mnemonic="ADDSUBPS" detail="xmmreg2 to xmmreg1"
>
        <arg direction="input" type="xmmreg" />
        <arg direction="output" type="xmmreg" />
        <description>Add /Sub packed SP FP numbers from XMM2/Mem to XMM1</description>
    </op>
    <op bitmask="0010000" mnemonic="AND" detail="register to memory">
        <arg direction="input" type="mem" />
        <arg direction="input" type="reg" />
        <arg direction="output" type="mem" />
        <description>Logical AND</description>
    </op>
    <op bitmask="0010000x11" mnemonic="AND" detail="register1 to register2">
        <arg direction="input" type="reg" />
        <arg direction="input" type="reg" />
        <arg direction="output" type="reg" />
        <description>Logical AND</description>
    </op>
    <op bitmask="0010001" mnemonic="AND" detail="memory to register">
        <arg direction="input" type="reg" />
        <arg direction="input" type="mem" />
        <arg direction="output" type="reg" />
        <description>Logical AND</description>
    </op>
    <op bitmask="0010001x11" mnemonic="AND" detail="register2 to register1">
        <arg direction="input" type="reg" />
        <arg direction="input" type="reg" />
        <arg direction="output" type="reg" />
        <description>Logical AND</description>
    </op>
    <op bitmask="0010010" mnemonic="AND" detail="immediate to AL, AX, or EAX">
        <arg direction="input" type="reg" />
        <arg direction="input" type="imm" />
        <arg direction="output" type="reg" />
        <description>Logical AND</description>
    </op>
    <op bitmask="100000xx11100" mnemonic="AND" detail="immediate to register">
        <arg direction="input" type="reg" />
        <arg direction="input" type="imm" />
        <arg direction="output" type="reg" />
        <description>Logical AND</description>
    </op>
    <op bitmask="100000xxxx100" mnemonic="AND" detail="immediate to memory">
```

```
        <arg direction="input" type="mem" />
        <arg direction="input" type="imm" />
        <arg direction="output" type="mem" />
        <description>Logical AND</description>
    </op>
    <op bitmask="011001100000111101010101" mnemonic="ANDNPD" detail="mem to xmmreg">
        <arg direction="input" type="xmmreg" />
        <arg direction="input" type="mem" />
        <arg direction="output" type="xmmreg" />
        <description>Bitwise Logical AND NOT of Packed Double-Precision Floating-Point Value
s</description>
    </op>
    <op bitmask="01100110000011110101010111" mnemonic="ANDNPD" detail="xmmreg to xmmreg">
        <arg direction="input" type="xmmreg" />
        <arg direction="input" type="xmmreg" />
        <arg direction="output" type="xmmreg" />
        <description>Bitwise Logical AND NOT of Packed Double-Precision Floating-Point Value
s</description>
    </op>
    <op bitmask="0000111101010101" mnemonic="ANDNPS" detail="mem to xmmreg">
        <arg direction="input" type="xmmreg" />
        <arg direction="input" type="mem" />
        <arg direction="output" type="xmmreg" />
        <description>Bitwise Logical AND NOT of Packed Single-Precision Floating-Point Value
s</description>
    </op>
    <op bitmask="000011110101010111" mnemonic="ANDNPS" detail="xmmreg to xmmreg">
        <arg direction="input" type="xmmreg" />
        <arg direction="input" type="xmmreg" />
        <arg direction="output" type="xmmreg" />
        <description>Bitwise Logical AND NOT of Packed Single-Precision Floating-Point Value
s</description>
    </op>
    <op bitmask="011001100000111101010100" mnemonic="ANDPD" detail="mem to xmmreg">
        <arg direction="input" type="xmmreg" />
        <arg direction="input" type="mem" />
        <arg direction="output" type="xmmreg" />
        <description>Bitwise Logical AND of Packed Double-Precision Floating-Point Values</d
escription>
    </op>
    <op bitmask="01100110000011110101010011" mnemonic="ANDPD" detail="xmmreg to xmmreg">
        <arg direction="input" type="xmmreg" />
        <arg direction="input" type="xmmreg" />
        <arg direction="output" type="xmmreg" />
        <description>Bitwise Logical AND of Packed Double-Precision Floating-Point Values</d
escription>
    </op>
    <op bitmask="0000111101010100" mnemonic="ANDPS" detail="mem to xmmreg">
        <arg direction="input" type="xmmreg" />
        <arg direction="input" type="mem" />
        <arg direction="output" type="xmmreg" />
        <description>Bitwise Logical AND of Packed Single-Precision Floating-Point Values</d
escription>
    </op>
    <op bitmask="000011110101010011" mnemonic="ANDPS" detail="xmmreg to xmmreg">
        <arg direction="input" type="xmmreg" />
        <arg direction="input" type="xmmreg" />
        <arg direction="output" type="xmmreg" />
        <description>Bitwise Logical AND of Packed Single-Precision Floating-Point Values</d
escription>
    </op>
    <op bitmask="01100011" mnemonic="ARPL" detail="from memory">
        <arg direction="input" type="mem" />
        <description>Adjust RPL Field of Selector</description>
    </op>
```

```
    <op bitmask="0110001111" mnemonic="ARPL" detail="from register">
        <arg direction="input" type="reg" />
        <description>Adjust RPL Field of Selector</description>
    </op>
    <op bitmask="01100010" mnemonic="BOUND">
        <arg direction="input" type="reg" />
        <arg direction="output" type="reg" />
        <description>Check Array Against Bounds</description>
    </op>
    <op bitmask="0000111110111100" mnemonic="BSF" detail="memory, register">
        <arg direction="input" type="reg" />
        <arg direction="output" type="mem" />
        <description>Bit Scan Forward</description>
    </op>
    <op bitmask="0000111111011110011" mnemonic="BSF" detail="register1, register2">
        <arg direction="input" type="reg" />
        <arg direction="output" type="reg" />
        <description>Bit Scan Forward</description>
    </op>
    <op bitmask="0000111110111101" mnemonic="BSR" detail="memory, register">
        <arg direction="input" type="reg" />
        <arg direction="output" type="mem" />
        <description>Bit Scan Reverse</description>
    </op>
    <op bitmask="0000111111011110111" mnemonic="BSR" detail="register1, register2">
        <arg direction="input" type="reg" />
        <arg direction="output" type="reg" />
        <description>Bit Scan Reverse</description>
    </op>
    <op bitmask="000011111001" mnemonic="BSWAP">
        <arg direction="input" type="reg" />
        <arg direction="output" type="reg" />
        <description>Byte Swap</description>
    </op>
    <op bitmask="0000111110100011" mnemonic="BT" detail="memory, reg">
        <arg direction="input" type="mem" />
        <arg direction="input" type="reg" />
        <description>Bit Test</description>
    </op>
    <op bitmask="0000111111010001111" mnemonic="BT" detail="register1, register2">
        <arg direction="input" type="reg" />
        <arg direction="input" type="reg" />
        <description>Bit Test</description>
    </op>
    <op bitmask="00001111110111010111100" mnemonic="BT" detail="register, immediate">
        <arg direction="input" type="reg" />
        <arg direction="input" type="imm" />
        <description>Bit Test</description>
    </op>
    <op bitmask="0000111110111010xx100" mnemonic="BT" detail="memory, immediate">
        <arg direction="input" type="mem" />
        <arg direction="input" type="imm" />
        <description>Bit Test</description>
    </op>
    <op bitmask="00001111110111010111111" mnemonic="BTC" detail="register, immediate">
        <arg direction="input" type="reg" />
        <arg direction="input" type="imm" />
        <description>Bit Test and Complement</description>
    </op>
    <op bitmask="0000111110111010xx111" mnemonic="BTC" detail="memory, immediate">
        <arg direction="input" type="mem" />
        <arg direction="input" type="imm" />
        <description>Bit Test and Complement</description>
    </op>
    <op bitmask="0000111110111011" mnemonic="BTC" detail="memory, reg">
```

```
        <arg direction="input" type="mem" />
        <arg direction="input" type="reg" />
        <description>Bit Test and Complement</description>
    </op>
    <op bitmask="0000111111011101111" mnemonic="BTC" detail="register1, register2">
        <arg direction="input" type="reg" />
        <arg direction="input" type="reg" />
        <description>Bit Test and Complement</description>
    </op>
    <op bitmask="0000111110110011" mnemonic="BTR" detail="memory, reg">
        <arg direction="input" type="mem" />
        <arg direction="input" type="reg" />
        <description>Bit Test and Reset</description>
    </op>
    <op bitmask="0000111111011001111" mnemonic="BTR" detail="register1, register2">
        <arg direction="input" type="reg" />
        <arg direction="input" type="reg" />
        <description>Bit Test and Reset</description>
    </op>
    <op bitmask="00001111110111010111110" mnemonic="BTR" detail="register, immediate">
        <arg direction="input" type="reg" />
        <arg direction="input" type="imm" />
        <description>Bit Test and Reset</description>
    </op>
    <op bitmask="0000111110111010xx110" mnemonic="BTR" detail="memory, immediate">
        <arg direction="input" type="mem" />
        <arg direction="input" type="imm" />
        <description>Bit Test and Reset</description>
    </op>
    <op bitmask="0000111110101011" mnemonic="BTS" detail="memory, reg">
        <arg direction="input" type="mem" />
        <arg direction="input" type="reg" />
        <description>Bit Test and Set</description>
    </op>
    <op bitmask="0000111111010101111" mnemonic="BTS" detail="register1, register2">
        <arg direction="input" type="reg" />
        <arg direction="input" type="reg" />
        <description>Bit Test and Set</description>
    </op>
    <op bitmask="00001111110111010111101" mnemonic="BTS" detail="register, immediate">
        <arg direction="input" type="reg" />
        <arg direction="input" type="imm" />
        <description>Bit Test and Set</description>
    </op>
    <op bitmask="0000111110111010xx101" mnemonic="BTS" detail="memory, immediate">
        <arg direction="input" type="mem" />
        <arg direction="input" type="imm" />
        <description>Bit Test and Set</description>
    </op>
    <op bitmask="10011010" mnemonic="CALL" detail="direct">
        <arg direction="input" type="imm" />
        <description>Call Procedure (in other segment)</description>
    </op>
    <op bitmask="11101000" mnemonic="CALL" detail="direct">
        <arg direction="input" type="imm" />
        <description>Call Procedure (in same segment)</description>
    </op>
    <op bitmask="1111111111010" mnemonic="CALL" detail="register indirect">
        <arg direction="input" type="reg" />
        <description>Call Procedure (in same segment)</description>
    </op>
    <op bitmask="11111111xx010" mnemonic="CALL" detail="memory indirect">
        <arg direction="input" type="mem" />
        <description>Call Procedure (in same segment)</description>
    </op>
```

```
    <op bitmask="11111111xx011" mnemonic="CALL" detail="indirect">
        <arg direction="input" type="reg" />
        <description>Call Procedure (in other segment)</description>
    </op>
    <op bitmask="10011000" mnemonic="CBW/CWDE">
        <description>Convert Word to Doubleword</description>
    </op>
    <op bitmask="10011001" mnemonic="CDQ/CWD">
        <description>Convert Word to Doubleword</description>
    </op>
    <op bitmask="11111000" mnemonic="CLC">
        <description>Clear Carry Flag</description>
    </op>
    <op bitmask="11111100" mnemonic="CLD">
        <description>Clear Direction Flag</description>
    </op>
    <op bitmask="0000111110101110" mnemonic="CLFLUSH" detail="mem">
        <arg direction="input" type="mem" />
        <arg direction="output" type="mem" />
        <description>Flush Cache Line</description>
    </op>
    <op bitmask="11111010" mnemonic="CLI">
        <description>Clear Interrupt Flag</description>
    </op>
    <op bitmask="0000111100000110" mnemonic="CLTS">
        <description>Clear Task-Switched Flag in CR0</description>
    </op>
    <op bitmask="11110101" mnemonic="CMC">
        <description>Complement Carry Flag</description>
    </op>
    <op bitmask="0000111101000000" mnemonic="CMOVcc" detail="memory to register" conditional
="O">
        <arg direction="input" type="mem" />
        <arg direction="output" type="reg" />
        <description>Conditional Move: Overflow</description>
    </op>
    <op bitmask="0000111100000011" mnemonic="CMOVcc" detail="register2 to register1" condi
tional="O">
        <arg direction="input" type="reg" />
        <arg direction="output" type="reg" />
        <description>Conditional Move: Overflow</description>
    </op>
    <op bitmask="0000111101000001" mnemonic="CMOVcc" detail="memory to register" conditional
="NO">
        <arg direction="input" type="mem" />
        <arg direction="output" type="reg" />
        <description>Conditional Move: No overflow</description>
    </op>
    <op bitmask="0000111100000111" mnemonic="CMOVcc" detail="register2 to register1" condi
tional="NO">
        <arg direction="input" type="reg" />
        <arg direction="output" type="reg" />
        <description>Conditional Move: No overflow</description>
    </op>
    <op bitmask="0000111101000010" mnemonic="CMOVcc" detail="memory to register" conditional
="B">
        <arg direction="input" type="mem" />
        <arg direction="output" type="reg" />
        <description>Conditional Move: Below</description>
    </op>
    <op bitmask="0000111100001011" mnemonic="CMOVcc" detail="register2 to register1" condi
tional="B">
        <arg direction="input" type="reg" />
        <arg direction="output" type="reg" />
        <description>Conditional Move: Below</description>
```

```
    </op>
    <op bitmask="0000111101000011" mnemonic="CMOVcc" detail="memory to register" conditional
="NB">
        <arg direction="input" type="mem" />
        <arg direction="output" type="reg" />
        <description>Conditional Move: Not below</description>
    </op>
    <op bitmask="0000111100001111" mnemonic="CMOVcc" detail="register2 to register1" condi
tional="NB">
        <arg direction="input" type="reg" />
        <arg direction="output" type="reg" />
        <description>Conditional Move: Not below</description>
    </op>
    <op bitmask="0000111101000100" mnemonic="CMOVcc" detail="memory to register" conditional
="E">
        <arg direction="input" type="mem" />
        <arg direction="output" type="reg" />
        <description>Conditional Move: Equals</description>
    </op>
    <op bitmask="0000111100010011" mnemonic="CMOVcc" detail="register2 to register1" condi
tional="E">
        <arg direction="input" type="reg" />
        <arg direction="output" type="reg" />
        <description>Conditional Move: Equals</description>
    </op>
    <op bitmask="0000111101000101" mnemonic="CMOVcc" detail="memory to register" conditional
="NE">
        <arg direction="input" type="mem" />
        <arg direction="output" type="reg" />
        <description>Conditional Move: Not equals</description>
    </op>
    <op bitmask="0000111100010111" mnemonic="CMOVcc" detail="register2 to register1" condi
tional="NE">
        <arg direction="input" type="reg" />
        <arg direction="output" type="reg" />
        <description>Conditional Move: Not equals</description>
    </op>
    <op bitmask="0000111101000110" mnemonic="CMOVcc" detail="memory to register" conditional
="NA">
        <arg direction="input" type="mem" />
        <arg direction="output" type="reg" />
        <description>Conditional Move: Not above</description>
    </op>
    <op bitmask="0000111100011011" mnemonic="CMOVcc" detail="register2 to register1" condi
tional="NA">
        <arg direction="input" type="reg" />
        <arg direction="output" type="reg" />
        <description>Conditional Move: Not above</description>
    </op>
    <op bitmask="0000111101000111" mnemonic="CMOVcc" detail="memory to register" conditional
="A">
        <arg direction="input" type="mem" />
        <arg direction="output" type="reg" />
        <description>Conditional Move: Above</description>
    </op>
    <op bitmask="0000111100011111" mnemonic="CMOVcc" detail="register2 to register1" condi
tional="A">
        <arg direction="input" type="reg" />
        <arg direction="output" type="reg" />
        <description>Conditional Move: Above</description>
    </op>
    <op bitmask="0000111101001000" mnemonic="CMOVcc" detail="memory to register" conditional
="S">
        <arg direction="input" type="mem" />
        <arg direction="output" type="reg" />
```

```
        <description>Conditional Move: Sign</description>
    </op>
    <op bitmask="000011110100100011" mnemonic="CMOVcc" detail="register2 to register1" condi
tional="S">
        <arg direction="input" type="reg" />
        <arg direction="output" type="reg" />
        <description>Conditional Move: Sign</description>
    </op>
    <op bitmask="0000111101001001" mnemonic="CMOVcc" detail="memory to register" conditional
="NS">
        <arg direction="input" type="mem" />
        <arg direction="output" type="reg" />
        <description>Conditional Move: Not sign</description>
    </op>
    <op bitmask="000011110100100111" mnemonic="CMOVcc" detail="register2 to register1" condi
tional="NS">
        <arg direction="input" type="reg" />
        <arg direction="output" type="reg" />
        <description>Conditional Move: Not sign</description>
    </op>
    <op bitmask="0000111101001010" mnemonic="CMOVcc" detail="memory to register" conditional
="P">
        <arg direction="input" type="mem" />
        <arg direction="output" type="reg" />
        <description>Conditional Move: Parity</description>
    </op>
    <op bitmask="000011110100101011" mnemonic="CMOVcc" detail="register2 to register1" condi
tional="P">
        <arg direction="input" type="reg" />
        <arg direction="output" type="reg" />
        <description>Conditional Move: Parity</description>
    </op>
    <op bitmask="0000111101001011" mnemonic="CMOVcc" detail="memory to register" conditional
="NP">
        <arg direction="input" type="mem" />
        <arg direction="output" type="reg" />
        <description>Conditional Move: Not parity</description>
    </op>
    <op bitmask="000011110100101111" mnemonic="CMOVcc" detail="register2 to register1" condi
tional="NP">
        <arg direction="input" type="reg" />
        <arg direction="output" type="reg" />
        <description>Conditional Move: Not parity</description>
    </op>
    <op bitmask="0000111101001100" mnemonic="CMOVcc" detail="memory to register" conditional
="L">
        <arg direction="input" type="mem" />
        <arg direction="output" type="reg" />
        <description>Conditional Move: Less than</description>
    </op>
    <op bitmask="000011110100110011" mnemonic="CMOVcc" detail="register2 to register1" condi
tional="L">
        <arg direction="input" type="reg" />
        <arg direction="output" type="reg" />
        <description>Conditional Move: Less than</description>
    </op>
    <op bitmask="0000111101001101" mnemonic="CMOVcc" detail="memory to register" conditional
="NL">
        <arg direction="input" type="mem" />
        <arg direction="output" type="reg" />
        <description>Conditional Move: Not less than</description>
    </op>
    <op bitmask="000011110100110111" mnemonic="CMOVcc" detail="register2 to register1" condi
tional="NL">
        <arg direction="input" type="reg" />
```

```
        <arg direction="output" type="reg" />
        <description>Conditional Move: Not less than</description>
    </op>
    <op bitmask="0000111101001110" mnemonic="CMOVcc" detail="memory to register" conditional
="NG">
        <arg direction="input" type="mem" />
        <arg direction="output" type="reg" />
        <description>Conditional Move: Not greater than</description>
    </op>
    <op bitmask="000011110100111011" mnemonic="CMOVcc" detail="register2 to register1" condi
tional="NG">
        <arg direction="input" type="reg" />
        <arg direction="output" type="reg" />
        <description>Conditional Move: Not greater than</description>
    </op>
    <op bitmask="0000111101001111" mnemonic="CMOVcc" detail="memory to register" conditional
="G">
        <arg direction="input" type="mem" />
        <arg direction="output" type="reg" />
        <description>Conditional Move: Greater than</description>
    </op>
    <op bitmask="000011110100111111" mnemonic="CMOVcc" detail="register2 to register1" condi
tional="G">
        <arg direction="input" type="reg" />
        <arg direction="output" type="reg" />
        <description>Conditional Move: Greater than</description>
    </op>
    <op bitmask="0011100" mnemonic="CMP" detail="memory with register">
        <arg direction="input" type="mem" />
        <arg direction="input" type="reg" />
        <description>Compare Two Operands</description>
    </op>
    <op bitmask="0011100x11" mnemonic="CMP" detail="register1 with register2">
        <arg direction="input" type="reg" />
        <arg direction="input" type="reg" />
        <description>Compare Two Operands</description>
    </op>
    <op bitmask="0011101" mnemonic="CMP" detail="register with memory">
        <arg direction="input" type="reg" />
        <arg direction="input" type="mem" />
        <description>Compare Two Operands</description>
    </op>
    <op bitmask="0011101x11" mnemonic="CMP" detail="register2 with register1">
        <arg direction="input" type="reg" />
        <arg direction="input" type="reg" />
        <description>Compare Two Operands</description>
    </op>
    <op bitmask="0011110" mnemonic="CMP" detail="immediate with AL, AX, or EAX">
        <arg direction="input" type="imm" />
        <arg direction="input" type="reg" />
        <description>Compare Two Operands</description>
    </op>
    <op bitmask="100000xx11111" mnemonic="CMP" detail="immediate with register">
        <arg direction="input" type="imm" />
        <arg direction="input" type="reg" />
        <description>Compare Two Operands</description>
    </op>
    <op bitmask="100000xxxx111" mnemonic="CMP" detail="immediate with memory">
        <arg direction="input" type="imm" />
        <arg direction="input" type="mem" />
        <description>Compare Two Operands</description>
    </op>
    <op bitmask="011001100000111111000010" mnemonic="CMPPD" detail="mem to xmmreg, imm8">
        <arg direction="input" type="mem" />
        <arg direction="input" type="xmmreg" />
```

```
        <arg direction="input" type="imm" />
        <description>Compare Packed Double-Precision Floating-Point Values imm8</description
>
    </op>
    <op bitmask="01100110000011111100001011" mnemonic="CMPPD" detail="xmmreg to xmmreg, imm8
">
        <arg direction="input" type="xmmreg" />
        <arg direction="input" type="xmmreg" />
        <arg direction="input" type="imm" />
        <description>Compare Packed Double-Precision Floating-Point Values imm8</description
>
    </op>
    <op bitmask="0000111111000010" mnemonic="CMPPS" detail="mem to xmmreg, imm8">
        <arg direction="input" type="mem" />
        <arg direction="input" type="xmmreg" />
        <arg direction="input" type="imm" />
        <description>Compare Packed Single-Precision Floating-Point Values</description>
    </op>
    <op bitmask="00001111111000001011" mnemonic="CMPPS" detail="xmmreg to xmmreg, imm8">
        <arg direction="input" type="xmmreg" />
        <arg direction="input" type="xmmreg" />
        <arg direction="input" type="imm" />
        <description>Compare Packed Single-Precision Floating-Point Values</description>
    </op>
    <op bitmask="1010011" mnemonic="CMPS/CMPSB/CMPSW/CMPSD">
        <description>Compare String Operands</description>
    </op>
    <op bitmask="11110010000011111111000010" mnemonic="CMPSD" detail="mem to xmmreg, imm8">
        <arg direction="input" type="mem" />
        <arg direction="input" type="xmmreg" />
        <arg direction="input" type="imm" />
        <description>Compare Scalar Double-Precision Floating-Point Values imm8</description
>
    </op>
    <op bitmask="11110010000011111100001011" mnemonic="CMPSD" detail="xmmreg to xmmreg, imm8
">
        <arg direction="input" type="xmmreg" />
        <arg direction="input" type="xmmreg" />
        <arg direction="input" type="imm" />
        <description>Compare Scalar Double-Precision Floating-Point Values imm8</description
>
    </op>
    <op bitmask="11110011000011111111000010" mnemonic="CMPSS" detail="mem to xmmreg, imm8">
        <arg direction="input" type="mem" />
        <arg direction="input" type="xmmreg" />
        <arg direction="input" type="imm" />
        <description>Compare Scalar Single-Precision Floating-Point Values imm8</description
>
    </op>
    <op bitmask="11110011000011111100001011" mnemonic="CMPSS" detail="xmmreg to xmmreg, imm8
">
        <arg direction="input" type="xmmreg" />
        <arg direction="input" type="xmmreg" />
        <arg direction="input" type="imm" />
        <description>Compare Scalar Single-Precision Floating-Point Values imm8</description
>
    </op>
    <op bitmask="0000111111011000" mnemonic="CMPXCHG" detail="memory, register">
        <arg direction="input" type="mem" />
        <arg direction="input" type="reg" />
        <arg direction="output" type="mem" />
        <arg direction="output" type="reg" />
        <description>Compare and Exchange</description>
    </op>
    <op bitmask="0000111111011000x11" mnemonic="CMPXCHG" detail="register1, register2">
```

```
        <arg direction="input" type="reg" />
        <arg direction="input" type="reg" />
        <arg direction="output" type="reg" />
        <arg direction="output" type="reg" />
        <description>Compare and Exchange</description>
    </op>
    <op bitmask="0000111111000111xx001" mnemonic="CMPXCHG8B" detail="memory, register">
        <arg direction="input" type="mem" />
        <arg direction="input" type="reg" />
        <arg direction="output" type="mem" />
        <arg direction="output" type="reg" />
        <description>Compare and Exchange 8 Bytes</description>
    </op>
    <op bitmask="011001100000111100101111" mnemonic="COMISD" detail="mem to xmmreg">
        <arg direction="input" type="mem" />
        <arg direction="input" type="xmmreg" />
        <description>Compare Scalar Ordered Double-Precision Floating-Point Values and Set E
FLAGS</description>
    </op>
    <op bitmask="0110011000001111100101111" mnemonic="COMISD" detail="xmmreg to xmmreg">
        <arg direction="input" type="xmmreg" />
        <arg direction="input" type="xmmreg" />
        <description>Compare Scalar Ordered Double-Precision Floating-Point Values and Set E
FLAGS</description>
    </op>
    <op bitmask="0000111100101111" mnemonic="COMISS" detail="mem to xmmreg">
        <arg direction="input" type="mem" />
        <arg direction="input" type="xmmreg" />
        <description>Compare Scalar Ordered Single-Precision Floating-Point Values and Set E
FLAGS</description>
    </op>
    <op bitmask="0000111100101111" mnemonic="COMISS" detail="xmmreg to xmmreg">
        <arg direction="input" type="xmmreg" />
        <arg direction="input" type="xmmreg" />
        <description>Compare Scalar Ordered Single-Precision Floating-Point Values and Set E
FLAGS</description>
    </op>
    <op bitmask="0000111110100010" mnemonic="CPUID">
        <description>CPU Identification</description>
    </op>
    <op bitmask="111100110000111111100110" mnemonic="CVTDQ2PD" detail="mem to xmmreg">
        <arg direction="input" type="mem" />
        <arg direction="output" type="xmmreg" />
        <description>Convert Packed Doubleword Integers to Packed Single-Precision Floating-
Point Values</description>
    </op>
    <op bitmask="11110011000011111110011011" mnemonic="CVTDQ2PD" detail="xmmreg to xmmreg">
        <arg direction="input" type="xmmreg" />
        <arg direction="output" type="xmmreg" />
        <description>Convert Packed Doubleword Integers to Packed Single-Precision Floating-
Point Values</description>
    </op>
    <op bitmask="0000111101011011" mnemonic="CVTDQ2PS" detail="mem to xmmreg">
        <arg direction="input" type="mem" />
        <arg direction="output" type="xmmreg" />
        <description>Convert Packed Doubleword Integers to Packed Double-Precision Floating-
Point Values</description>
    </op>
    <op bitmask="0000111110101101111" mnemonic="CVTDQ2PS" detail="xmmreg to xmmreg">
        <arg direction="input" type="xmmreg" />
        <arg direction="output" type="xmmreg" />
        <description>Convert Packed Doubleword Integers to Packed Double-Precision Floating-
Point Values</description>
    </op>
    <op bitmask="111100100000111111100110" mnemonic="CVTPD2DQ" detail="mem to xmmreg">
```

```
        <arg direction="input" type="mem" />
        <arg direction="output" type="xmmreg" />
        <description>Convert Packed Double-Precision Floating-Point Values to Packed Doublew
ord Integers</description>
    </op>
    <op bitmask="11110010000011111110011011" mnemonic="CVTPD2DQ" detail="xmmreg to xmmreg">
        <arg direction="input" type="xmmreg" />
        <arg direction="output" type="xmmreg" />
        <description>Convert Packed Double-Precision Floating-Point Values to Packed Doublew
ord Integers</description>
    </op>
    <op bitmask="011001100000111100101101" mnemonic="CVTPD2PI" detail="mem to mmreg">
        <arg direction="input" type="mem" />
        <arg direction="output" type="mmreg" />
        <description>Convert Packed Double-Precision Floating-Point Values to Packed Doublew
ord Integers</description>
    </op>
    <op bitmask="011001100000111110010110111" mnemonic="CVTPD2PI" detail="xmmreg to mmreg">
        <arg direction="input" type="xmmreg" />
        <arg direction="output" type="mmreg" />
        <description>Convert Packed Double-Precision Floating-Point Values to Packed Doublew
ord Integers</description>
    </op>
    <op bitmask="011001100000111101011010" mnemonic="CVTPD2PS" detail="mem to xmmreg">
        <arg direction="input" type="mem" />
        <arg direction="output" type="xmmreg" />
        <description>Covert Packed Double-Precision Floating-Point Values to Packed Single-P
recision Floating-Point Values</description>
    </op>
    <op bitmask="011001100000111110101101011" mnemonic="CVTPD2PS" detail="xmmreg to xmmreg">
        <arg direction="input" type="xmmreg" />
        <arg direction="output" type="xmmreg" />
        <description>Covert Packed Double-Precision Floating-Point Values to Packed Single-P
recision Floating-Point Values</description>
    </op>
    <op bitmask="011001100000111100101010" mnemonic="CVTPI2PD" detail="mem to xmmreg">
        <arg direction="input" type="mem" />
        <arg direction="output" type="xmmreg" />
        <description>Convert Packed Doubleword Integers to Packed Double-Precision Floating-
Point Values</description>
    </op>
    <op bitmask="011001100000111110010101011" mnemonic="CVTPI2PD" detail="mmreg to xmmreg">
        <arg direction="input" type="mmreg" />
        <arg direction="output" type="xmmreg" />
        <description>Convert Packed Doubleword Integers to Packed Double-Precision Floating-
Point Values</description>
    </op>
    <op bitmask="0000111100101010" mnemonic="CVTPI2PS" detail="mem to xmmreg">
        <arg direction="input" type="mem" />
        <arg direction="output" type="xmmreg" />
        <description>Convert Packed Doubleword Integers to Packed Single-Precision Floating-
Point Values</description>
    </op>
    <op bitmask="0000111110010101011" mnemonic="CVTPI2PS" detail="mmreg to xmmreg">
        <arg direction="input" type="mmreg" />
        <arg direction="output" type="xmmreg" />
        <description>Convert Packed Doubleword Integers to Packed Single-Precision Floating-
Point Values</description>
    </op>
    <op bitmask="011001100000111101011011" mnemonic="CVTPS2DQ" detail="mem to xmmreg">
        <arg direction="input" type="mem" />
        <arg direction="output" type="xmmreg" />
        <description>Convert Packed Single-Precision Floating-Point Values to Packed Doublew
ord Integers</description>
    </op>
```

```
    <op bitmask="01100110000011110101101111" mnemonic="CVTPS2DQ" detail="xmmreg to xmmreg">
        <arg direction="input" type="xmmreg" />
        <arg direction="output" type="xmmreg" />
        <description>Convert Packed Single-Precision Floating-Point Values to Packed Doublew
ord Integers</description>
    </op>
    <op bitmask="0000111101011010" mnemonic="CVTPS2PD" detail="mem to xmmreg">
        <arg direction="input" type="mem" />
        <arg direction="output" type="xmmreg" />
        <description>Covert Packed Single-Precision Floating-Point Values to Packed Double-P
recision Floating-Point Values</description>
    </op>
    <op bitmask="0000111101011011" mnemonic="CVTPS2PD" detail="xmmreg to xmmreg">
        <arg direction="input" type="xmmreg" />
        <arg direction="output" type="xmmreg" />
        <description>Covert Packed Single-Precision Floating-Point Values to Packed Double-P
recision Floating-Point Values</description>
    </op>
    <op bitmask="0000111100101101" mnemonic="CVTPS2PI" detail="mem to mmreg">
        <arg direction="input" type="mem" />
        <arg direction="output" type="mmreg" />
        <description>Convert Packed Single-Precision Floating-Point Values to Packed Doublew
ord Integers</description>
    </op>
    <op bitmask="0000111100010110111" mnemonic="CVTPS2PI" detail="xmmreg to mmreg">
        <arg direction="input" type="xmmreg" />
        <arg direction="output" type="mmreg" />
        <description>Convert Packed Single-Precision Floating-Point Values to Packed Doublew
ord Integers</description>
    </op>
    <op bitmask="11110010000011110010101101" mnemonic="CVTSD2SI" detail="mem to r32">
        <arg direction="input" type="mem" />
        <arg direction="output" type="reg" />
        <description>Convert Scalar Double-Precision Floating-Point Value to Doubleword Inte
ger</description>
    </op>
    <op bitmask="11110010000011110010110111" mnemonic="CVTSD2SI" detail="xmmreg to r32">
        <arg direction="input" type="xmmreg" />
        <arg direction="output" type="reg" />
        <description>Convert Scalar Double-Precision Floating-Point Value to Doubleword Inte
ger</description>
    </op>
    <op bitmask="11110010000011110101011010" mnemonic="CVTSD2SS" detail="mem to xmmreg">
        <arg direction="input" type="mem" />
        <arg direction="output" type="xmmreg" />
        <description>Covert Scalar Double-Precision Floating-Point Value to Scalar Single-Pr
ecision Floating-Point Value</description>
    </op>
    <op bitmask="11110010000011110101101011" mnemonic="CVTSD2SS" detail="xmmreg to xmmreg">
        <arg direction="input" type="xmmreg" />
        <arg direction="output" type="xmmreg" />
        <description>Covert Scalar Double-Precision Floating-Point Value to Scalar Single-Pr
ecision Floating-Point Value</description>
    </op>
    <op bitmask="11110010000011110010101010" mnemonic="CVTSI2SD" detail="mem to xmmreg">
        <arg direction="input" type="mem" />
        <arg direction="output" type="xmmreg" />
        <description>Convert Doubleword Integer to Scalar Double-Precision Floating-Point Va
lue</description>
    </op>
    <op bitmask="11110010000011110010101011" mnemonic="CVTSI2SD" detail="r32 to xmmreg1">
        <arg direction="input" type="reg" />
        <arg direction="output" type="xmmreg" />
        <description>Convert Doubleword Integer to Scalar Double-Precision Floating-Point Va
lue</description>
```

```
    </op>
    <op bitmask="11110011000001111100101010" mnemonic="CVTSI2SS" detail="mem to xmmreg">
        <arg direction="input" type="mem" />
        <arg direction="output" type="xmmreg" />
        <description>Convert Doubleword Integer to Scalar Single-Precision Floating-Point Va
lue</description>
    </op>
    <op bitmask="11110011000001111110010101011" mnemonic="CVTSI2SS" detail="r32 to xmmreg1">
        <arg direction="input" type="reg" />
        <arg direction="output" type="xmmreg" />
        <description>Convert Doubleword Integer to Scalar Single-Precision Floating-Point Va
lue</description>
    </op>
    <op bitmask="11110011000001111101011010" mnemonic="CVTSS2SD" detail="mem to xmmreg">
        <arg direction="input" type="mem" />
        <arg direction="output" type="xmmreg" />
        <description>Covert Scalar Single-Precision Floating-Point Value to Scalar Double-Pr
ecision Floating-Point Value</description>
    </op>
    <op bitmask="11110011000001111101101011" mnemonic="CVTSS2SD" detail="xmmreg to xmmreg">
        <arg direction="input" type="xmmreg" />
        <arg direction="output" type="xmmreg" />
        <description>Covert Scalar Single-Precision Floating-Point Value to Scalar Double-Pr
ecision Floating-Point Value</description>
    </op>
    <op bitmask="11110011000001111100101101" mnemonic="CVTSS2SI" detail="mem to r32">
        <arg direction="input" type="mem" />
        <arg direction="output" type="reg" />
        <description>Convert Scalar Single-Precision Floating-Point Value to Doubleword Inte
ger</description>
    </op>
    <op bitmask="11110011000001111110010110111" mnemonic="CVTSS2SI" detail="xmmreg to r32">
        <arg direction="input" type="xmmreg" />
        <arg direction="output" type="reg" />
        <description>Convert Scalar Single-Precision Floating-Point Value to Doubleword Inte
ger</description>
    </op>
    <op bitmask="01100110000001111111100110" mnemonic="CVTTPD2DQ" detail="mem to xmmreg">
        <arg direction="input" type="mem" />
        <arg direction="output" type="xmmreg" />
        <description>Convert With Truncation Packed Double-Precision Floating-Point Values t
o Packed Doubleword Integers</description>
    </op>
    <op bitmask="01100110000001111111110011011" mnemonic="CVTTPD2DQ" detail="xmmreg to xmmreg">
        <arg direction="input" type="xmmreg" />
        <arg direction="output" type="xmmreg" />
        <description>Convert With Truncation Packed Double-Precision Floating-Point Values t
o Packed Doubleword Integers</description>
    </op>
    <op bitmask="01100110000001111100101100" mnemonic="CVTTPD2PI" detail="mem to mmreg">
        <arg direction="input" type="mem" />
        <arg direction="output" type="mmreg" />
        <description>Convert with Truncation Packed Double-Precision Floating-Point Values t
o Packed Doubleword Integers</description>
    </op>
    <op bitmask="01100110000001111110010110011" mnemonic="CVTTPD2PI" detail="xmmreg to mmreg">
        <arg direction="input" type="xmmreg" />
        <arg direction="output" type="mmreg" />
        <description>Convert with Truncation Packed Double-Precision Floating-Point Values t
o Packed Doubleword Integers</description>
    </op>
    <op bitmask="11110011000001111101011011" mnemonic="CVTTPS2DQ" detail="mem to xmmreg">
        <arg direction="input" type="mem" />
        <arg direction="output" type="xmmreg" />
        <description>Convert With Truncation Packed Single-Precision Floating-Point Values t
```

```
o Packed Doubleword Integers</description>
    </op>
    <op bitmask="11110011000001111010110111l" mnemonic="CVTTPS2DQ" detail="xmmreg to xmmreg">
        <arg direction="input" type="xmmreg" />
        <arg direction="output" type="xmmreg" />
        <description>Convert With Truncation Packed Single-Precision Floating-Point Values t
o Packed Doubleword Integers</description>
    </op>
    <op bitmask="0000111100101100" mnemonic="CVTTPS2PI" detail="mem to mmreg">
        <arg direction="input" type="mem" />
        <arg direction="output" type="mmreg" />
        <description>Convert with Truncation Packed Single-Precision Floating-Point Values t
o Packed Doubleword Integers</description>
    </op>
    <op bitmask="000011110010110011" mnemonic="CVTTPS2PI" detail="xmmreg to mmreg">
        <arg direction="input" type="xmmreg" />
        <arg direction="output" type="mmreg" />
        <description>Convert with Truncation Packed Single-Precision Floating-Point Values t
o Packed Doubleword Integers</description>
    </op>
    <op bitmask="11110010000011110010110011" mnemonic="CVTTSD2SI" detail="mem to r32">
        <arg direction="input" type="mem" />
        <arg direction="output" type="reg" />
        <description>Convert with Truncation Scalar Double-Precision Floating-Point Value to
 Doubleword Integer</description>
    </op>
    <op bitmask="11110010000011110010110011" mnemonic="CVTTSD2SI" detail="xmmreg to r32">
        <arg direction="input" type="xmmreg" />
        <arg direction="output" type="reg" />
        <description>Convert with Truncation Scalar Double-Precision Floating-Point Value to
 Doubleword Integer</description>
    </op>
    <op bitmask="11110011000011110010110011" mnemonic="CVTTSS2SI" detail="mem to r32">
        <arg direction="input" type="mem" />
        <arg direction="output" type="reg" />
        <description>Convert with Truncation Scalar Single-Precision Floating-Point Value to
 Doubleword Integer</description>
    </op>
    <op bitmask="11110011000011110010110011" mnemonic="CVTTSS2SI" detail="xmmreg to r32">
        <arg direction="input" type="xmmreg" />
        <arg direction="output" type="reg" />
        <description>Convert with Truncation Scalar Single-Precision Floating-Point Value to
 Doubleword Integer</description>
    </op>
    <op bitmask="00100111" mnemonic="DAA">
        <description>Decimal Adjust AL after Addition</description>
    </op>
    <op bitmask="00101111" mnemonic="DAS">
        <description>Decimal Adjust AL after Subtraction</description>
    </op>
    <op bitmask="01001" mnemonic="DEC" detail="register (alternate encoding)">
        <arg direction="input" type="reg" />
        <arg direction="output" type="reg" />
        <description>Decrement by 1</description>
    </op>
    <op bitmask="1111111x11001" mnemonic="DEC" detail="register">
        <arg direction="input" type="reg" />
        <arg direction="output" type="reg" />
        <description>Decrement by 1</description>
    </op>
    <op bitmask="1111111xxx001" mnemonic="DEC" detail="memory">
        <arg direction="input" type="mem" />
        <arg direction="output" type="mem" />
        <description>Decrement by 1</description>
    </op>
```

```
    <op bitmask="1111011x11110" mnemonic="DIV" detail="AL, AX, or EAX by register">
        <arg direction="input" type="reg" />
        <arg direction="input" type="reg" />
        <arg direction="output" type="reg" />
        <description>Unsigned Divide</description>
    </op>
    <op bitmask="1111011xxx110" mnemonic="DIV" detail="AL, AX, or EAX by memory">
        <arg direction="input" type="reg" />
        <arg direction="input" type="mem" />
        <arg direction="output" type="reg" />
        <description>Unsigned Divide</description>
    </op>
    <op bitmask="01100110000011110101111O" mnemonic="DIVPD" detail="mem to xmmreg">
        <arg direction="input" type="mem" />
        <arg direction="output" type="xmmreg" />
        <description>Divide Packed Double-Precision Floating-Point Values</description>
    </op>
    <op bitmask="0110011000001111010111O1" mnemonic="DIVPD" detail="xmmreg to xmmreg">
        <arg direction="input" type="xmmreg" />
        <arg direction="output" type="xmmreg" />
        <description>Divide Packed Double-Precision Floating-Point Values</description>
    </op>
    <op bitmask="0000111101011110" mnemonic="DIVPS" detail="mem to xmmreg">
        <arg direction="input" type="mem" />
        <arg direction="output" type="xmmreg" />
        <description>Divide Packed Single-Precision Floating-Point Values</description>
    </op>
    <op bitmask="000011110101111O" mnemonic="DIVPS" detail="xmmreg to xmmreg">
        <arg direction="input" type="xmmreg" />
        <arg direction="output" type="xmmreg" />
        <description>Divide Packed Single-Precision Floating-Point Values</description>
    </op>
    <op bitmask="11110010000011110101111O" mnemonic="DIVSD" detail="mem to xmmreg">
        <arg direction="input" type="mem" />
        <arg direction="output" type="xmmreg" />
        <description>Divide Scalar Double-Precision Floating-Point Values</description>
    </op>
    <op bitmask="1111001000001111010111O1" mnemonic="DIVSD" detail="xmmreg to xmmreg">
        <arg direction="input" type="xmmreg" />
        <arg direction="output" type="xmmreg" />
        <description>Divide Scalar Double-Precision Floating-Point Values</description>
    </op>
    <op bitmask="11110011000011110101111O" mnemonic="DIVSS" detail="mem to xmmreg">
        <arg direction="input" type="mem" />
        <arg direction="output" type="xmmreg" />
        <description>Divide Scalar Single-Precision Floating-Point Values</description>
    </op>
    <op bitmask="1111001100001111010111O1" mnemonic="DIVSS" detail="xmmreg to xmmreg">
        <arg direction="input" type="xmmreg" />
        <arg direction="output" type="xmmreg" />
        <description>Divide Scalar Single-Precision Floating-Point Values</description>
    </op>
    <op bitmask="0000111101110111" mnemonic="EMMS">
        <description>Empty MMX technology state</description>
    </op>
    <op bitmask="11001000" mnemonic="ENTER">
        <arg direction="input" type="imm" />
        <arg direction="input" type="imm" />
        <description>Make Stack Frame for High Level Procedure</description>
    </op>
    <op bitmask="1101100111110000" mnemonic="F2XM1">
        <description>Compute 2^ST(0) -1</description>
    </op>
    <op bitmask="1101100111100001" mnemonic="FABS">
        <description>Absolute Value</description>
```

```
    </op>
    <op bitmask="11011000xx000" mnemonic="FADD" detail="ST(0) = ST(0) + 32-bit memory">
        <arg direction="input" type="ST(i)" />
        <arg direction="input" type="mem" />
        <arg direction="output" type="ST(i)" />
        <description>Add</description>
    </op>
    <op bitmask="11011100xx000" mnemonic="FADD" detail="ST(0) = ST(0) + 64-bit memory">
        <arg direction="input" type="ST(i)" />
        <arg direction="input" type="mem" />
        <arg direction="output" type="ST(i)" />
        <description>Add</description>
    </op>
    <op bitmask="1101111000" mnemonic="FADD" detail="ST(d) = ST(0) + ST(i)">
        <arg direction="input" type="ST(i)" />
        <arg direction="input" type="ST(i)" />
        <arg direction="output" type="ST(i)" />
        <description>Add</description>
    </op>
    <op bitmask="1101111011000" mnemonic="FADDP" detail="ST(0) = ST(0) + ST(i)">
        <arg direction="input" type="ST(i)" />
        <arg direction="input" type="ST(i)" />
        <arg direction="output" type="ST(i)" />
        <description>Add and Pop</description>
    </op>
    <op bitmask="11011111xx100" mnemonic="FBLD">
        <description>Load Binary Coded Decimal</description>
    </op>
    <op bitmask="11011111xx110" mnemonic="FBSTP">
        <description>Store Binary Coded Decimal and Pop</description>
    </op>
    <op bitmask="1101100111100000" mnemonic="FCHS">
        <description>Change Sign</description>
    </op>
    <op bitmask="1101101111100010" mnemonic="FCLEX">
        <description>Clear Exceptions</description>
    </op>
    <op bitmask="1101101011000" mnemonic="FCMOVcc" detail="move if below (B)" conditional="B
">
        <arg direction="input" type="ST(i)" />
        <description>Conditional Move on EFLAG Register Condition Codes: move if below (B)</
description>
    </op>
    <op bitmask="1101101011001" mnemonic="FCMOVcc" detail="move if equal (E)" conditional="E
">
        <arg direction="input" type="ST(i)" />
        <description>Conditional Move on EFLAG Register Condition Codes: move if equal (E)</
description>
    </op>
    <op bitmask="1101101011010" mnemonic="FCMOVcc" detail="move if below or equal (BE)" cond
itional="BE">
        <arg direction="input" type="ST(i)" />
        <description>Conditional Move on EFLAG Register Condition Codes: move if below or eq
ual (BE)</description>
    </op>
    <op bitmask="1101101011011" mnemonic="FCMOVcc" detail="move if unordered (U)" conditiona
l="U">
        <arg direction="input" type="ST(i)" />
        <description>Conditional Move on EFLAG Register Condition Codes: move if unordered (
U)</description>
    </op>
    <op bitmask="1101101111000" mnemonic="FCMOVcc" detail="move if not below (NB)" condition
al="NB">
        <arg direction="input" type="ST(i)" />
        <description>Conditional Move on EFLAG Register Condition Codes: move if not below (
```

```
NB)</description>
    </op>
    <op bitmask="1101101111001" mnemonic="FCMOVcc" detail="move if not equal (NE)" condition
al="NE">
        <arg direction="input" type="ST(i)" />
        <description>Conditional Move on EFLAG Register Condition Codes: move if not equal (
NE)</description>
    </op>
    <op bitmask="1101101111010" mnemonic="FCMOVcc" detail="move if not below or equal (NBE)"
 conditional="NBE">
        <arg direction="input" type="ST(i)" />
        <description>Conditional Move on EFLAG Register Condition Codes: move if not below o
r equal (NBE)</description>
    </op>
    <op bitmask="1101101111011" mnemonic="FCMOVcc" detail="move if not unordered (NU)" condi
tional="NU">
        <arg direction="input" type="ST(i)" />
        <description>Conditional Move on EFLAG Register Condition Codes: move if not unorder
ed (NU)</description>
    </op>
    <op bitmask="1101100011010" mnemonic="FCOM" detail="ST(i)">
        <arg direction="input" type="ST(i)" />
        <description>Compare Real</description>
    </op>
    <op bitmask="11011000xx010" mnemonic="FCOM" detail="32-bit memory">
        <arg direction="input" type="mem" />
        <description>Compare Real</description>
    </op>
    <op bitmask="11011100xx010" mnemonic="FCOM" detail="64-bit memory">
        <arg direction="input" type="mem" />
        <description>Compare Real</description>
    </op>
    <op bitmask="1101101111110" mnemonic="FCOMI">
        <arg direction="input" type="ST(i)" />
        <description>Compare Real and Set EFLAGS</description>
    </op>
    <op bitmask="1101111111110" mnemonic="FCOMIP">
        <arg direction="input" type="ST(i)" />
        <description>Compare Real, Set EFLAGS, and Pop</description>
    </op>
    <op bitmask="1101100011011" mnemonic="FCOMP" detail="ST(i)">
        <arg direction="input" type="ST(i)" />
        <description>Compare Real and Pop</description>
    </op>
    <op bitmask="11011000xx011" mnemonic="FCOMP" detail="32-bit memory">
        <arg direction="input" type="mem" />
        <description>Compare Real and Pop</description>
    </op>
    <op bitmask="11011100xx011" mnemonic="FCOMP" detail="64-bit memory">
        <arg direction="input" type="mem" />
        <description>Compare Real and Pop</description>
    </op>
    <op bitmask="1101111011011001" mnemonic="FCOMPP">
        <description>Compare Real and Pop Twice</description>
    </op>
    <op bitmask="1101100111111111" mnemonic="FCOS">
        <description>Cosine of ST(0)</description>
    </op>
    <op bitmask="1101100111110110" mnemonic="FDECSTP">
        <description>Decrement Stack-Top Pointer</description>
    </op>
    <op bitmask="11011000xx110" mnemonic="FDIV" detail="ST(0) = ST(0) / 32-bit memory">
        <arg direction="input" type="ST(i)" />
        <arg direction="input" type="mem" />
        <arg direction="output" type="ST(i)" />
```

```
        <description>Divide</description>
    </op>
    <op bitmask="11011100xx110" mnemonic="FDIV" detail="ST(0) = ST(0) / 64-bit memory">
        <arg direction="input" type="ST(i)" />
        <arg direction="input" type="mem" />
        <arg direction="output" type="ST(i)" />
        <description>Divide</description>
    </op>
    <op bitmask="110111111" mnemonic="FDIV/FDIVR" detail="ST(d) = ST(0) / ST(i), ST(d) = ST(
i) / ST(0)">
        <arg direction="input" type="ST(i)" />
        <arg direction="input" type="ST(i)" />
        <arg direction="output" type="ST(i)" />
        <description>Reverse Divide</description>
    </op>
    <op bitmask="1101111011111" mnemonic="FDIVP" detail="ST(0) = ST(0) / ST(i)">
        <arg direction="input" type="ST(i)" />
        <arg direction="input" type="ST(i)" />
        <arg direction="output" type="ST(i)" />
        <description>Divide and Pop</description>
    </op>
    <op bitmask="11011000xx111" mnemonic="FDIVR" detail="ST(0) = 32-bit memory / ST(0)">
        <arg direction="input" type="mem" />
        <arg direction="input" type="ST(i)" />
        <arg direction="output" type="ST(i)" />
        <description>Reverse Divide</description>
    </op>
    <op bitmask="11011100xx111" mnemonic="FDIVR" detail="ST(0) = 64-bit memory / ST(0)">
        <arg direction="input" type="mem" />
        <arg direction="input" type="ST(i)" />
        <arg direction="output" type="ST(i)" />
        <description>Reverse Divide</description>
    </op>
    <op bitmask="1101111011110" mnemonic="FDIVRP" detail="ST(0) = ST(i) / ST(0)">
        <arg direction="input" type="ST(i)" />
        <arg direction="input" type="ST(i)" />
        <arg direction="output" type="ST(i)" />
        <description>Reverse Divide and Pop</description>
    </op>
    <op bitmask="1101110111000" mnemonic="FFREE">
        <arg direction="input" type="ST(i)" />
        <arg direction="output" type="ST(i)" />
        <description>Free ST(i) Register</description>
    </op>
    <op bitmask="11011010xx000" mnemonic="FIADD" detail="ST(0) = ST(0) + 32-bit memory">
        <arg direction="input" type="ST(i)" />
        <arg direction="input" type="mem" />
        <arg direction="output" type="ST(i)" />
        <description>Add Integer</description>
    </op>
    <op bitmask="11011110xx000" mnemonic="FIADD" detail="ST(0) = ST(0) + 16-bit memory">
        <arg direction="input" type="ST(i)" />
        <arg direction="input" type="mem" />
        <arg direction="output" type="ST(i)" />
        <description>Add Integer</description>
    </op>
    <op bitmask="11011010xx010" mnemonic="FICOM" detail="32-bit memory">
        <arg direction="input" type="mem" />
        <description>Compare Integer</description>
    </op>
    <op bitmask="11011110xx010" mnemonic="FICOM" detail="16-bit memory">
        <arg direction="input" type="mem" />
        <description>Compare Integer</description>
    </op>
    <op bitmask="11011010xx011" mnemonic="FICOMP" detail="32-bit memory">
```

```
        <arg direction="input" type="mem" />
        <description>Compare Integer and Pop</description>
    </op>
    <op bitmask="11011110xx011" mnemonic="FICOMP" detail="16-bit memory">
        <arg direction="input" type="mem" />
        <description>Compare Integer and Pop</description>
    </op>
    <op bitmask="11011010xx110" mnemonic="FIDIV" detail="ST(0) = ST(0) / 32-bit memory">
        <arg direction="input" type="ST(i)" />
        <arg direction="input" type="mem" />
        <arg direction="output" type="ST(i)" />
        <description>FIDIV</description>
    </op>
    <op bitmask="11011110xx110" mnemonic="FIDIV" detail="ST(0) = ST(0) / 16-bit memory">
        <arg direction="input" type="ST(i)" />
        <arg direction="input" type="mem" />
        <arg direction="output" type="ST(i)" />
        <description>FIDIV</description>
    </op>
    <op bitmask="11011010xx111" mnemonic="FIDIVR" detail="ST(0) = 32-bit memory / ST(0)">
        <arg direction="input" type="mem" />
        <arg direction="input" type="ST(i)" />
        <arg direction="output" type="ST(i)" />
        <description>FIDIVR</description>
    </op>
    <op bitmask="11011110xx111" mnemonic="FIDIVR" detail="ST(0) = 16-bit memory / ST(0)">
        <arg direction="input" type="mem" />
        <arg direction="input" type="ST(i)" />
        <arg direction="output" type="ST(i)" />
        <description>FIDIVR</description>
    </op>
    <op bitmask="11011011xx000" mnemonic="FILD" detail="32-bit memory">
        <arg direction="input" type="mem" />
        <arg direction="output" type="ST(i)" />
        <description>Load Integer</description>
    </op>
    <op bitmask="11011111xx000" mnemonic="FILD" detail="16-bit memory">
        <arg direction="input" type="mem" />
        <arg direction="output" type="ST(i)" />
        <description>Load Integer</description>
    </op>
    <op bitmask="11011111xx101" mnemonic="FILD" detail="64-bit memory">
        <arg direction="input" type="mem" />
        <arg direction="output" type="ST(i)" />
        <description>Load Integer</description>
    </op>
    <op bitmask="11011010xx001" mnemonic="FIMUL" detail="ST(0) = ST(0) * 32-bit memory">
        <arg direction="input" type="ST(i)" />
        <arg direction="input" type="mem" />
        <arg direction="output" type="ST(i)" />
        <description>FIMUL</description>
    </op>
    <op bitmask="11011110xx001" mnemonic="FIMUL" detail="ST(0) = ST(0) * 16-bit memory">
        <arg direction="input" type="ST(i)" />
        <arg direction="input" type="mem" />
        <arg direction="output" type="ST(i)" />
        <description>FIMUL</description>
    </op>
    <op bitmask="1101100111110111" mnemonic="FINCSTP">
        <description>Increment Stack Pointer</description>
    </op>
    <op bitmask="1101101111100011" mnemonic="FINIT">
        <description>Initialize FPU</description>
    </op>
    <op bitmask="11011011xx010" mnemonic="FIST" detail="32-bit memory">
```

```
        <arg direction="input" type="mem" />
        <arg direction="output" type="ST(i)" />
        <description>Store Integer</description>
    </op>
    <op bitmask="11011111xx010" mnemonic="FIST" detail="16-bit memory">
        <arg direction="input" type="mem" />
        <arg direction="output" type="ST(i)" />
        <description>Store Integer</description>
    </op>
    <op bitmask="11011011xx011" mnemonic="FISTP" detail="32-bit memory">
        <arg direction="input" type="mem" />
        <arg direction="output" type="ST(i)" />
        <description>Store Integer and Pop</description>
    </op>
    <op bitmask="11011111xx011" mnemonic="FISTP" detail="16-bit memory">
        <arg direction="input" type="mem" />
        <arg direction="output" type="ST(i)" />
        <description>Store Integer and Pop</description>
    </op>
    <op bitmask="11011111xx111" mnemonic="FISTP" detail="64-bit memory">
        <arg direction="input" type="mem" />
        <arg direction="output" type="ST(i)" />
        <description>Store Integer and Pop</description>
    </op>
    <op bitmask="11011011xx001" mnemonic="FISTTP" detail="m32int">
        <arg direction="input" type="mem" />
        <arg direction="output" type="mem" />
        <description>Store ST in int32 (chop) and pop</description>
    </op>
    <op bitmask="11011101xx001" mnemonic="FISTTP" detail="m64int">
        <arg direction="input" type="mem" />
        <arg direction="output" type="mem" />
        <description>Store ST in int64 (chop) and pop</description>
    </op>
    <op bitmask="11011111xx001" mnemonic="FISTTP" detail="m16int">
        <arg direction="input" type="mem" />
        <arg direction="output" type="mem" />
        <description>Store ST in int16 (chop) and pop</description>
    </op>
    <op bitmask="11011010xx100" mnemonic="FISUB" detail="ST(0) = ST(0) - 32-bit memory">
        <arg direction="input" type="ST(i)" />
        <arg direction="input" type="mem" />
        <arg direction="output" type="ST(i)" />
        <description>FISUB</description>
    </op>
    <op bitmask="11011110xx100" mnemonic="FISUB" detail="ST(0) = ST(0) - 16-bit memory">
        <arg direction="input" type="ST(i)" />
        <arg direction="input" type="mem" />
        <arg direction="output" type="ST(i)" />
        <description>FISUB</description>
    </op>
    <op bitmask="11011010xx101" mnemonic="FISUBR" detail="ST(0) = 32-bit memory - ST(0)">
        <arg direction="input" type="mem" />
        <arg direction="input" type="ST(i)" />
        <arg direction="output" type="ST(i)" />
        <description>FISUBR</description>
    </op>
    <op bitmask="11011110xx101" mnemonic="FISUBR" detail="ST(0) = 16-bit memory - ST(0)">
        <arg direction="input" type="mem" />
        <arg direction="input" type="ST(i)" />
        <arg direction="output" type="ST(i)" />
        <description>FISUBR</description>
    </op>
    <op bitmask="1101100111000" mnemonic="FLD" detail="ST(i)">
        <arg direction="input" type="ST(i)" />
```

```
        <arg direction="output" type="ST(i)" />
        <description>Load Real</description>
    </op>
    <op bitmask="11011001xx000" mnemonic="FLD" detail="32-bit memory">
        <arg direction="input" type="mem" />
        <arg direction="output" type="ST(i)" />
        <description>Load Real</description>
    </op>
    <op bitmask="11011011xx101" mnemonic="FLD" detail="80-bit memory">
        <arg direction="input" type="mem" />
        <arg direction="output" type="ST(i)" />
        <description>Load Real</description>
    </op>
    <op bitmask="11011101xx000" mnemonic="FLD" detail="64-bit memory">
        <arg direction="input" type="mem" />
        <arg direction="output" type="ST(i)" />
        <description>Load Real</description>
    </op>
    <op bitmask="1101100111101000" mnemonic="FLD1">
        <arg direction="output" type="ST(0)" />
        <description>Load +1.0 into ST(0)</description>
    </op>
    <op bitmask="11011001xx101" mnemonic="FLDCW">
        <description>Load Control Word</description>
    </op>
    <op bitmask="11011001xx100" mnemonic="FLDENV">
        <description>Load FPU Environment</description>
    </op>
    <op bitmask="1101100111101010" mnemonic="FLDL2E">
        <arg direction="output" type="ST(0)" />
        <description>Load log2() into ST(0)</description>
    </op>
    <op bitmask="1101100111101001" mnemonic="FLDL2T">
        <arg direction="output" type="ST(0)" />
        <description>Load log2(10) into ST(0)</description>
    </op>
    <op bitmask="1101100111101100" mnemonic="FLDLG2">
        <arg direction="output" type="ST(0)" />
        <description>Load log10(2) into ST(0)</description>
    </op>
    <op bitmask="1101100111101101" mnemonic="FLDLN2">
        <arg direction="output" type="ST(0)" />
        <description>Load log(2) into ST(0)</description>
    </op>
    <op bitmask="1101100111101011" mnemonic="FLDPI">
        <arg direction="output" type="ST(0)" />
        <description>Load into ST(0)</description>
    </op>
    <op bitmask="1101100111101110" mnemonic="FLDZ">
        <arg direction="output" type="ST(0)" />
        <description>Load +0.0 into ST(0)</description>
    </op>
    <op bitmask="11011000xx001" mnemonic="FMUL" detail="ST(0) = ST(0) * 32-bit memory">
        <arg direction="input" type="ST(i)" />
        <arg direction="input" type="mem" />
        <arg direction="output" type="ST(i)" />
        <description>Multiply</description>
    </op>
    <op bitmask="11011100xx001" mnemonic="FMUL" detail="ST(0) = ST(0) * 64-bit memory">
        <arg direction="input" type="ST(i)" />
        <arg direction="input" type="mem" />
        <arg direction="output" type="ST(i)" />
        <description>Multiply</description>
    </op>
    <op bitmask="1101111001" mnemonic="FMUL" detail="ST(d) = ST(0) * ST(i)">
```

```
        <arg direction="input" type="ST(i)" />
        <arg direction="input" type="ST(i)" />
        <arg direction="output" type="ST(i)" />
        <description>Multiply</description>
    </op>
    <op bitmask="1101111011001" mnemonic="FMULP" detail="ST(i) = ST(0) * ST(i)">
        <arg direction="input" type="ST(i)" />
        <arg direction="input" type="ST(i)" />
        <arg direction="output" type="ST(i)" />
        <description>Multiply</description>
    </op>
    <op bitmask="1101100111010000" mnemonic="FNOP">
        <description>No Operation</description>
    </op>
    <op bitmask="1101100111110011" mnemonic="FPATAN">
        <description>Partial Arctangent</description>
    </op>
    <op bitmask="1101100111111000" mnemonic="FPREM">
        <description>Partial Remainder</description>
    </op>
    <op bitmask="1101100111110101" mnemonic="FPREM1">
        <description>Partial Remainder (IEEE)</description>
    </op>
    <op bitmask="1101100111110010" mnemonic="FPTAN">
        <description>Partial Tangent</description>
    </op>
    <op bitmask="1101100111111100" mnemonic="FRNDINT">
        <description>Round to Integer</description>
    </op>
    <op bitmask="11011101xx100" mnemonic="FRSTOR">
        <description>Restore FPU State</description>
    </op>
    <op bitmask="11011101xx110" mnemonic="FSAVE">
        <description>Store FPU State</description>
    </op>
    <op bitmask="1101100111111101" mnemonic="FSCALE">
        <description>Scale</description>
    </op>
    <op bitmask="1101100111111110" mnemonic="FSIN">
        <description>Sine</description>
    </op>
    <op bitmask="1101100111111011" mnemonic="FSINCOS">
        <description>Sine and Cosine</description>
    </op>
    <op bitmask="1101100111111010" mnemonic="FSQRT">
        <description>Square Root</description>
    </op>
    <op bitmask="11011001xx010" mnemonic="FST" detail="32-bit memory">
        <arg direction="input" type="mem" />
        <arg direction="output" type="ST(i)" />
        <description>Store Real</description>
    </op>
    <op bitmask="1101110111010" mnemonic="FST" detail="ST(i)">
        <arg direction="input" type="ST(i)" />
        <arg direction="output" type="ST(i)" />
        <description>Store Real</description>
    </op>
    <op bitmask="11011101xx010" mnemonic="FST" detail="64-bit memory">
        <arg direction="input" type="mem" />
        <arg direction="output" type="ST(i)" />
        <description>Store Real</description>
    </op>
    <op bitmask="11011001xx111" mnemonic="FSTCW">
        <description>Store Control Word</description>
    </op>
```

```
    <op bitmask="11011001xx110" mnemonic="FSTENV">
        <description>Store FPU Environment</description>
    </op>
    <op bitmask="11011001xx011" mnemonic="FSTP" detail="32-bit memory">
        <arg direction="input" type="mem" />
        <arg direction="output" type="ST(i)" />
        <description>Store Real and Pop</description>
    </op>
    <op bitmask="11011011xx111" mnemonic="FSTP" detail="80-bit memory">
        <arg direction="input" type="mem" />
        <arg direction="output" type="ST(i)" />
        <description>Store Real and Pop</description>
    </op>
    <op bitmask="1101110111011" mnemonic="FSTP" detail="ST(i)">
        <arg direction="input" type="ST(i)" />
        <arg direction="output" type="ST(i)" />
        <description>Store Real and Pop</description>
    </op>
    <op bitmask="11011101xx011" mnemonic="FSTP" detail="64-bit memory">
        <arg direction="input" type="mem" />
        <arg direction="output" type="ST(i)" />
        <description>Store Real and Pop</description>
    </op>
    <op bitmask="11011101xx111" mnemonic="FSTSW">
        <arg direction="output" type="Memory" />
        <description>Store Status Word into Memory</description>
    </op>
    <op bitmask="1101111111100000" mnemonic="FSTSW">
        <arg direction="output" type="AX" />
        <description>Store Status Word into AX</description>
    </op>
    <op bitmask="11011000xx100" mnemonic="FSUB" detail="ST(0) = ST(0) - 32-bit memory">
        <arg direction="input" type="ST(i)" />
        <arg direction="input" type="mem" />
        <arg direction="output" type="ST(i)" />
        <description>Subtract</description>
    </op>
    <op bitmask="11011100xx100" mnemonic="FSUB" detail="ST(0) = ST(0) - 64-bit memory">
        <arg direction="input" type="ST(i)" />
        <arg direction="input" type="mem" />
        <arg direction="output" type="ST(i)" />
        <description>Subtract</description>
    </op>
    <op bitmask="110111110" mnemonic="FSUB/FSUBR" detail="ST(d) = ST(0) - ST(i), ST(d) = ST(
i) - ST(0)">
        <arg direction="input" type="ST(i)" />
        <arg direction="input" type="ST(i)" />
        <arg direction="output" type="ST(i)" />
        <description>Reverse Subtract</description>
    </op>
    <op bitmask="1101111011101" mnemonic="FSUBP" detail="ST(0) = ST(0) - ST(i)">
        <arg direction="input" type="ST(i)" />
        <arg direction="input" type="ST(i)" />
        <arg direction="output" type="ST(i)" />
        <description>Subtract and Pop</description>
    </op>
    <op bitmask="11011000xx101" mnemonic="FSUBR" detail="ST(0) = 32-bit memory - ST(0)">
        <arg direction="input" type="mem" />
        <arg direction="input" type="ST(i)" />
        <arg direction="output" type="ST(i)" />
        <description>Reverse Subtract</description>
    </op>
    <op bitmask="11011100xx101" mnemonic="FSUBR" detail="ST(0) = 64-bit memory - ST(0)">
        <arg direction="input" type="mem" />
        <arg direction="input" type="ST(i)" />
```

```
        <arg direction="output" type="ST(i)" />
        <description>Reverse Subtract</description>
    </op>
    <op bitmask="1101111011100" mnemonic="FSUBRP" detail="ST(i) = ST(i) - ST(0)">
        <arg direction="input" type="ST(i)" />
        <arg direction="input" type="ST(i)" />
        <arg direction="output" type="ST(i)" />
        <description>Reverse Subtract and Pop</description>
    </op>
    <op bitmask="1101100111100100" mnemonic="FTST">
        <description>Test</description>
    </op>
    <op bitmask="1101110111100" mnemonic="FUCOM">
        <arg direction="input" type="ST(i)" />
        <description>Unordered Compare Real</description>
    </op>
    <op bitmask="1101101111101" mnemonic="FUCOMI">
        <arg direction="input" type="ST(i)" />
        <description>Unorderd Compare Real and Set EFLAGS</description>
    </op>
    <op bitmask="1101111111101" mnemonic="FUCOMIP">
        <arg direction="input" type="ST(i)" />
        <description>Unorderd Compare Real, Set EFLAGS, and Pop</description>
    </op>
    <op bitmask="1101110111101" mnemonic="FUCOMP">
        <arg direction="input" type="ST(i)" />
        <description>Unordered Compare Real and Pop</description>
    </op>
    <op bitmask="1101101011101001" mnemonic="FUCOMPP">
        <description>Unordered Compare Real and Pop Twice</description>
    </op>
    <op bitmask="1101100111100101" mnemonic="FXAM">
        <description>Examine</description>
    </op>
    <op bitmask="1101100111001" mnemonic="FXCH">
        <arg direction="input" type="ST(i)" />
        <arg direction="output" type="ST(i)" />
        <description>Exchange ST(0) and ST(i)</description>
    </op>
    <op bitmask="00001111101011110xx001" mnemonic="FXRSTOR">
        <description>Restore x87 FPU, MMX, SSE, and SSE2 State</description>
    </op>
    <op bitmask="00001111101011110xx000" mnemonic="FXSAVE">
        <description>Save x87 FPU, MMX, SSE, and SSE2 State</description>
    </op>
    <op bitmask="1101100111110100" mnemonic="FXTRACT">
        <description>Extract Exponent and Significand</description>
    </op>
    <op bitmask="1101100111110001" mnemonic="FYL2X">
        <description>ST(1) * log2(ST(0))</description>
    </op>
    <op bitmask="1101100111111001" mnemonic="FYL2XP1">
        <description>ST(1) * log2(ST(0) + 1.0)</description>
    </op>
    <op bitmask="011001100000111101111100" mnemonic="HADDPD" detail="mem to xmmreg">
        <arg direction="input" type="mem" />
        <arg direction="output" type="xmmreg" />
        <description>Add horizontally packed DP FP numbers XMM2/Mem to XMM1</description>
    </op>
    <op bitmask="01100110000011110111110011" mnemonic="HADDPD" detail="xmmreg2 to xmmreg1">
        <arg direction="input" type="xmmreg" />
        <arg direction="output" type="xmmreg" />
        <description>Add horizontally packed DP FP numbers XMM2/Mem to XMM1</description>
    </op>
    <op bitmask="111100100000111101111100" mnemonic="HADDPS" detail="mem to xmmreg">
```

```
        <arg direction="input" type="mem" />
        <arg direction="output" type="xmmreg" />
        <description>Add horizontally packed SP FP numbers XMM2/Mem to XMM1</description>
    </op>
    <op bitmask="11110010000011110111110011" mnemonic="HADDPS" detail="xmmreg2 to xmmreg1">
        <arg direction="input" type="xmmreg" />
        <arg direction="output" type="xmmreg" />
        <description>Add horizontally packed SP FP numbers XMM2/Mem to XMM1</description>
    </op>
    <op bitmask="11110100" mnemonic="HLT">
        <description>Halt</description>
    </op>
    <op bitmask="011001100000111101111101" mnemonic="HSUBPD" detail="mem to xmmreg">
        <arg direction="input" type="mem" />
        <arg direction="output" type="xmmreg" />
        <description>Sub horizontally packed DP FP numbers XMM2/Mem to XMM1</description>
    </op>
    <op bitmask="01100110000011110111110111" mnemonic="HSUBPD" detail="xmmreg2 to xmmreg1">
        <arg direction="input" type="xmmreg" />
        <arg direction="output" type="xmmreg" />
        <description>Sub horizontally packed DP FP numbers XMM2/Mem to XMM1</description>
    </op>
    <op bitmask="11110010000011110111101" mnemonic="HSUBPS" detail="mem to xmmreg">
        <arg direction="input" type="mem" />
        <arg direction="output" type="xmmreg" />
        <description>Sub horizontally packed SP FP numbers XMM2/Mem to XMM1</description>
    </op>
    <op bitmask="11110010000011110111110111" mnemonic="HSUBPS" detail="xmmreg2 to xmmreg1">
        <arg direction="input" type="xmmreg" />
        <arg direction="output" type="xmmreg" />
        <description>Sub horizontally packed SP FP numbers XMM2/Mem to XMM1</description>
    </op>
    <op bitmask="1111011x11111" mnemonic="IDIV" detail="AL, AX, or EAX by register">
        <arg direction="input" type="reg" />
        <arg direction="input" type="reg" />
        <arg direction="output" type="reg" />
        <description>Signed Divide</description>
    </op>
    <op bitmask="1111011xxx111" mnemonic="IDIV" detail="AL, AX, or EAX by memory">
        <arg direction="input" type="reg" />
        <arg direction="input" type="mem" />
        <arg direction="output" type="reg" />
        <description>Signed Divide</description>
    </op>
    <op bitmask="0000111110101111" mnemonic="IMUL" detail="register with memory">
        <arg direction="input" type="reg" />
        <arg direction="input" type="mem" />
        <arg direction="output" type="reg" />
        <description>Signed Multiply</description>
    </op>
    <op bitmask="000011111010111111" mnemonic="IMUL" detail="register1 with register2">
        <arg direction="input" type="reg" />
        <arg direction="input" type="reg" />
        <arg direction="output" type="reg" />
        <description>Signed Multiply</description>
    </op>
    <op bitmask="011010x1" mnemonic="IMUL" detail="memory with immediate to register">
        <arg direction="input" type="mem" />
        <arg direction="input" type="imm" />
        <arg direction="output" type="reg" />
        <description>Signed Multiply</description>
    </op>
    <op bitmask="011010x111" mnemonic="IMUL" detail="register1 with immediate to register2">
        <arg direction="input" type="reg" />
        <arg direction="input" type="imm" />
```

```
        <arg direction="output" type="reg" />
        <description>Signed Multiply</description>
    </op>
    <op bitmask="1111011x11101" mnemonic="IMUL" detail="AL, AX, or EAX with register">
        <arg direction="input" type="reg" />
        <arg direction="input" type="reg" />
        <arg direction="output" type="reg" />
        <description>Signed Multiply</description>
    </op>
    <op bitmask="1111011xxx101" mnemonic="IMUL" detail="AL, AX, or EAX with memory">
        <arg direction="input" type="reg" />
        <arg direction="input" type="mem" />
        <arg direction="output" type="reg" />
        <description>Signed Multiply</description>
    </op>
    <op bitmask="1110010" mnemonic="IN" detail="fixed port">
        <arg direction="input" type="imm" />
        <arg direction="output" type="reg" />
        <description>Input From Port</description>
    </op>
    <op bitmask="1110110" mnemonic="IN" detail="variable port">
        <arg direction="input" type="imm" />
        <arg direction="output" type="reg" />
        <description>Input From Port</description>
    </op>
    <op bitmask="01000" mnemonic="INC" detail="reg (alternate encoding)">
        <arg direction="input" type="reg" />
        <arg direction="output" type="reg" />
        <description>Increment by 1</description>
    </op>
    <op bitmask="1111111x11000" mnemonic="INC" detail="reg">
        <arg direction="input" type="reg" />
        <arg direction="output" type="reg" />
        <description>Increment by 1</description>
    </op>
    <op bitmask="1111111xxx000" mnemonic="INC" detail="memory">
        <arg direction="input" type="mem" />
        <arg direction="output" type="mem" />
        <description>Increment by 1</description>
    </op>
    <op bitmask="0110110" mnemonic="INS">
        <description>Input from DX Port</description>
    </op>
    <op bitmask="11001100" mnemonic="INT">
        <description>Single-Step Interrupt 3</description>
    </op>
    <op bitmask="11001101" mnemonic="INT n">
        <arg direction="input" type="imm" />
        <description>Interrupt Type n</description>
    </op>
    <op bitmask="11001110" mnemonic="INTO">
        <description>Interrupt 4 on Overflow</description>
    </op>
    <op bitmask="0000111100001000" mnemonic="INVD">
        <description>Invalidate Cache</description>
    </op>
    <op bitmask="0000111100000001xx111" mnemonic="INVLPG">
        <description>Invalidate TLB Entry</description>
    </op>
    <op bitmask="11001111" mnemonic="IRET/IRETD">
        <description>Interrupt Return</description>
    </op>
    <op bitmask="11100011" mnemonic="JCXZ/JECXZ">
        <arg direction="input" type="imm" />
        <description>Jump on CX/ECX Zero Address-size prefix differentiates JCXZ and JECXZ</
```

```
description>
    </op>
    <op bitmask="11101001" mnemonic="JMP" detail="direct">
        <arg direction="input" type="imm" />
        <description>Unconditional Jump (to same segment)</description>
    </op>
    <op bitmask="11101010" mnemonic="JMP" detail="direct intersegment">
        <arg direction="input" type="imm" />
        <description>Unconditional Jump (to other segment)</description>
    </op>
    <op bitmask="11101011" mnemonic="JMP" detail="short">
        <arg direction="input" type="imm" />
        <description>Unconditional Jump (to same segment)</description>
    </op>
    <op bitmask="1111111111100" mnemonic="JMP" detail="register indirect">
        <arg direction="input" type="reg" />
        <description>Unconditional Jump (to same segment)</description>
    </op>
    <op bitmask="11111111xx100" mnemonic="JMP" detail="memory indirect">
        <arg direction="input" type="mem" />
        <description>Unconditional Jump (to same segment)</description>
    </op>
    <op bitmask="11111111xx101" mnemonic="JMP" detail="indirect intersegment">
        <arg direction="input" type="reg" />
        <description>Unconditional Jump (to other segment)</description>
    </op>
    <op bitmask="0000111110000000" mnemonic="Jcc" detail="full displacement" conditional="O"
>
        <arg direction="input" type="imm" />
        <description>Jump if Condition is Met: Overflow</description>
    </op>
    <op bitmask="0000111110000001" mnemonic="Jcc" detail="full displacement" conditional="NO
">
        <arg direction="input" type="imm" />
        <description>Jump if Condition is Met: No overflow</description>
    </op>
    <op bitmask="0000111110000010" mnemonic="Jcc" detail="full displacement" conditional="B"
>
        <arg direction="input" type="imm" />
        <description>Jump if Condition is Met: Below</description>
    </op>
    <op bitmask="0000111110000011" mnemonic="Jcc" detail="full displacement" conditional="NB
">
        <arg direction="input" type="imm" />
        <description>Jump if Condition is Met: Not below</description>
    </op>
    <op bitmask="0000111110000100" mnemonic="Jcc" detail="full displacement" conditional="E"
>
        <arg direction="input" type="imm" />
        <description>Jump if Condition is Met: Equals</description>
    </op>
    <op bitmask="0000111110000101" mnemonic="Jcc" detail="full displacement" conditional="NE
">
        <arg direction="input" type="imm" />
        <description>Jump if Condition is Met: Not equals</description>
    </op>
    <op bitmask="0000111110000110" mnemonic="Jcc" detail="full displacement" conditional="NA
">
        <arg direction="input" type="imm" />
        <description>Jump if Condition is Met: Not above</description>
    </op>
    <op bitmask="0000111110000111" mnemonic="Jcc" detail="full displacement" conditional="A"
>
        <arg direction="input" type="imm" />
        <description>Jump if Condition is Met: Above</description>
```

```
    </op>
    <op bitmask="0000111110001000" mnemonic="Jcc" detail="full displacement" conditional="S"
>
        <arg direction="input" type="imm" />
        <description>Jump if Condition is Met: Sign</description>
    </op>
    <op bitmask="0000111110001001" mnemonic="Jcc" detail="full displacement" conditional="NS
">
        <arg direction="input" type="imm" />
        <description>Jump if Condition is Met: Not sign</description>
    </op>
    <op bitmask="0000111110001010" mnemonic="Jcc" detail="full displacement" conditional="P"
>
        <arg direction="input" type="imm" />
        <description>Jump if Condition is Met: Parity</description>
    </op>
    <op bitmask="0000111110001011" mnemonic="Jcc" detail="full displacement" conditional="NP
">
        <arg direction="input" type="imm" />
        <description>Jump if Condition is Met: Not parity</description>
    </op>
    <op bitmask="0000111110001100" mnemonic="Jcc" detail="full displacement" conditional="L"
>
        <arg direction="input" type="imm" />
        <description>Jump if Condition is Met: Less than</description>
    </op>
    <op bitmask="0000111110001101" mnemonic="Jcc" detail="full displacement" conditional="NL
">
        <arg direction="input" type="imm" />
        <description>Jump if Condition is Met: Not less than</description>
    </op>
    <op bitmask="0000111110001110" mnemonic="Jcc" detail="full displacement" conditional="NG
">
        <arg direction="input" type="imm" />
        <description>Jump if Condition is Met: Not greater than</description>
    </op>
    <op bitmask="0000111110001111" mnemonic="Jcc" detail="full displacement" conditional="G"
>
        <arg direction="input" type="imm" />
        <description>Jump if Condition is Met: Greater than</description>
    </op>
    <op bitmask="01110000" mnemonic="Jcc" detail="8-bit displacement" conditional="O">
        <arg direction="input" type="imm" />
        <description>Jump if Condition is Met: Overflow</description>
    </op>
    <op bitmask="01110001" mnemonic="Jcc" detail="8-bit displacement" conditional="NO">
        <arg direction="input" type="imm" />
        <description>Jump if Condition is Met: No overflow</description>
    </op>
    <op bitmask="01110010" mnemonic="Jcc" detail="8-bit displacement" conditional="B">
        <arg direction="input" type="imm" />
        <description>Jump if Condition is Met: Below</description>
    </op>
    <op bitmask="01110011" mnemonic="Jcc" detail="8-bit displacement" conditional="NB">
        <arg direction="input" type="imm" />
        <description>Jump if Condition is Met: Not below</description>
    </op>
    <op bitmask="01110100" mnemonic="Jcc" detail="8-bit displacement" conditional="E">
        <arg direction="input" type="imm" />
        <description>Jump if Condition is Met: Equals</description>
    </op>
    <op bitmask="01110101" mnemonic="Jcc" detail="8-bit displacement" conditional="NE">
        <arg direction="input" type="imm" />
        <description>Jump if Condition is Met: Not equals</description>
    </op>
```

```
    <op bitmask="01110110" mnemonic="Jcc" detail="8-bit displacement" conditional="NA">
        <arg direction="input" type="imm" />
        <description>Jump if Condition is Met: Not above</description>
    </op>
    <op bitmask="01110111" mnemonic="Jcc" detail="8-bit displacement" conditional="A">
        <arg direction="input" type="imm" />
        <description>Jump if Condition is Met: Above</description>
    </op>
    <op bitmask="01111000" mnemonic="Jcc" detail="8-bit displacement" conditional="S">
        <arg direction="input" type="imm" />
        <description>Jump if Condition is Met: Sign</description>
    </op>
    <op bitmask="01111001" mnemonic="Jcc" detail="8-bit displacement" conditional="NS">
        <arg direction="input" type="imm" />
        <description>Jump if Condition is Met: Not sign</description>
    </op>
    <op bitmask="01111010" mnemonic="Jcc" detail="8-bit displacement" conditional="P">
        <arg direction="input" type="imm" />
        <description>Jump if Condition is Met: Parity</description>
    </op>
    <op bitmask="01111011" mnemonic="Jcc" detail="8-bit displacement" conditional="NP">
        <arg direction="input" type="imm" />
        <description>Jump if Condition is Met: Not parity</description>
    </op>
    <op bitmask="01111100" mnemonic="Jcc" detail="8-bit displacement" conditional="L">
        <arg direction="input" type="imm" />
        <description>Jump if Condition is Met: Less than</description>
    </op>
    <op bitmask="01111101" mnemonic="Jcc" detail="8-bit displacement" conditional="NL">
        <arg direction="input" type="imm" />
        <description>Jump if Condition is Met: Not less than</description>
    </op>
    <op bitmask="01111110" mnemonic="Jcc" detail="8-bit displacement" conditional="NG">
        <arg direction="input" type="imm" />
        <description>Jump if Condition is Met: Not greater than</description>
    </op>
    <op bitmask="01111111" mnemonic="Jcc" detail="8-bit displacement" conditional="G">
        <arg direction="input" type="imm" />
        <description>Jump if Condition is Met: Greater than</description>
    </op>
    <op bitmask="10011111" mnemonic="LAHF">
        <arg direction="output" type="AHRegister" />
        <description>Load Flags into AHRegister</description>
    </op>
    <op bitmask="0000111100000010" mnemonic="LAR" detail="from memory">
        <arg direction="input" type="mem" />
        <description>Load Access Rights Byte</description>
    </op>
    <op bitmask="0000111100000011" mnemonic="LAR" detail="from register">
        <arg direction="input" type="reg" />
        <description>Load Access Rights Byte</description>
    </op>
    <op bitmask="111100100000111111110000" mnemonic="LDDQU" detail="xmm, m128">
        <arg direction="input" type="mem" />
        <arg direction="output" type="xmm" />
        <description>Load unaligned integer 128-bit</description>
    </op>
    <op bitmask="0000111110101110xx010" mnemonic="LDMXCSR" detail="m32 to MXCSR">
        <arg direction="input" type="mem" />
        <arg direction="output" type="MXCSR" />
        <description>Load MXCSR Register State</description>
    </op>
    <op bitmask="11000101" mnemonic="LDS">
        <arg direction="input" type="reg" />
        <arg direction="output" type="reg" />
```

```
        <description>Load Pointer to DS</description>
    </op>
    <op bitmask="10001101" mnemonic="LEA">
        <arg direction="input" type="reg" />
        <arg direction="output" type="reg" />
        <description>Load Effective Address</description>
    </op>
    <op bitmask="11001001" mnemonic="LEAVE">
        <description>High Level Procedure Exit</description>
    </op>
    <op bitmask="11000100" mnemonic="LES">
        <arg direction="input" type="reg" />
        <arg direction="output" type="reg" />
        <description>Load Pointer to ES</description>
    </op>
    <op bitmask="000011111010111011101000" mnemonic="LFENCE">
        <description>Load Fence</description>
    </op>
    <op bitmask="0000111110110100" mnemonic="LFS">
        <arg direction="input" type="reg" />
        <arg direction="output" type="reg" />
        <description>Load Pointer to FS</description>
    </op>
    <op bitmask="0000111100000001xx010" mnemonic="LGDT">
        <description>Load Global Descriptor Table Register</description>
    </op>
    <op bitmask="0000111110110101" mnemonic="LGS">
        <arg direction="input" type="reg" />
        <arg direction="output" type="reg" />
        <description>Load Pointer to GS</description>
    </op>
    <op bitmask="0000111100000001xx011" mnemonic="LIDT">
        <description>Load Interrupt Descriptor Table Register</description>
    </op>
    <op bitmask="0000111100000000011010" mnemonic="LLDT" detail="LDTR from register">
        <arg direction="input" type="reg" />
        <arg direction="output" type="LDTR" />
        <description>Load Local Descriptor Table Register</description>
    </op>
    <op bitmask="0000111100000000xx010" mnemonic="LLDT" detail="LDTR from memory">
        <arg direction="input" type="mem" />
        <arg direction="output" type="LDTR" />
        <description>Load Local Descriptor Table Register</description>
    </op>
    <op bitmask="0000111100000000111110" mnemonic="LMSW" detail="from register">
        <arg direction="input" type="reg" />
        <description>Load Machine Status Word</description>
    </op>
    <op bitmask="0000111100000001xx110" mnemonic="LMSW" detail="from memory">
        <arg direction="input" type="mem" />
        <description>Load Machine Status Word</description>
    </op>
    <op bitmask="11110000" mnemonic="LOCK">
        <description>Assert LOCK# Signal Prefix</description>
    </op>
    <op bitmask="1010110" mnemonic="LODS/LODSB/LODSW/LODSD">
        <description>Load String Operand</description>
    </op>
    <op bitmask="11100010" mnemonic="LOOP">
        <arg direction="input" type="imm" />
        <description>Loop Count</description>
    </op>
    <op bitmask="11100000" mnemonic="LOOPNZ/LOOPNE">
        <arg direction="input" type="imm" />
        <description>Loop Count while not Zero/Equal</description>
```

```
    </op>
    <op bitmask="11100001" mnemonic="LOOPZ/LOOPE">
        <arg direction="input" type="imm" />
        <description>Loop Count while Zero/Equal</description>
    </op>
    <op bitmask="0000111100000011" mnemonic="LSL" detail="from memory">
        <arg direction="input" type="mem" />
        <description>Load Segment Limit</description>
    </op>
    <op bitmask="0000111100000001111" mnemonic="LSL" detail="from register">
        <arg direction="input" type="reg" />
        <description>Load Segment Limit</description>
    </op>
    <op bitmask="0000111110110010" mnemonic="LSS">
        <arg direction="input" type="reg" />
        <arg direction="output" type="reg" />
        <description>Load Pointer to SS</description>
    </op>
    <op bitmask="00001111000000011011" mnemonic="LTR" detail="from register">
        <arg direction="input" type="reg" />
        <description>Load Task Register</description>
    </op>
    <op bitmask="0000111100000000xx011" mnemonic="LTR" detail="from memory">
        <arg direction="input" type="mem" />
        <description>Load Task Register</description>
    </op>
    <op bitmask="01100110000011111111011111" mnemonic="MASKMOVDQU" detail="xmmreg to xmmreg"
>
        <arg direction="input" type="xmmreg" />
        <arg direction="output" type="xmmreg" />
        <description>Store Selected Bytes of Double Quadword</description>
    </op>
    <op bitmask="0000111111111011111" mnemonic="MASKMOVQ" detail="mmreg to mmreg">
        <arg direction="input" type="mmreg" />
        <arg direction="output" type="mmreg" />
        <description>Store Selected Bytes of Quadword</description>
    </op>
    <op bitmask="01100110000011110101111" mnemonic="MAXPD" detail="mem to xmmreg">
        <arg direction="input" type="mem" />
        <arg direction="output" type="xmmreg" />
        <description>Return Maximum Packed Double-Precision Floating-Point Values</descripti
on>
    </op>
    <op bitmask="01100110000011110101111111" mnemonic="MAXPD" detail="xmmreg to xmmreg">
        <arg direction="input" type="xmmreg" />
        <arg direction="output" type="xmmreg" />
        <description>Return Maximum Packed Double-Precision Floating-Point Values</descripti
on>
    </op>
    <op bitmask="0000111101011111" mnemonic="MAXPS" detail="mem to xmmreg">
        <arg direction="input" type="mem" />
        <arg direction="output" type="xmmreg" />
        <description>Return Maximum Packed Single-Precision Floating-Point Values</descripti
on>
    </op>
    <op bitmask="00001111010111111" mnemonic="MAXPS" detail="xmmreg to xmmreg">
        <arg direction="input" type="xmmreg" />
        <arg direction="output" type="xmmreg" />
        <description>Return Maximum Packed Single-Precision Floating-Point Values</descripti
on>
    </op>
    <op bitmask="11110010000011110101111" mnemonic="MAXSD" detail="mem to xmmreg">
        <arg direction="input" type="mem" />
        <arg direction="output" type="xmmreg" />
        <description>Return Maximum Scalar Double-Precision Floating-Point Value</descriptio
```

```
n>
    </op>
    <op bitmask="11110010000011110101111111" mnemonic="MAXSD" detail="xmmreg to xmmreg">
        <arg direction="input" type="xmmreg" />
        <arg direction="output" type="xmmreg" />
        <description>Return Maximum Scalar Double-Precision Floating-Point Value</descriptio
n>
    </op>
    <op bitmask="11110011000011110101111" mnemonic="MAXSS" detail="mem to xmmreg">
        <arg direction="input" type="mem" />
        <arg direction="output" type="xmmreg" />
        <description>Return Maximum Scalar Double-Precision Floating-Point Value</descriptio
n>
    </op>
    <op bitmask="11110011000011110101111111" mnemonic="MAXSS" detail="xmmreg to xmmreg">
        <arg direction="input" type="xmmreg" />
        <arg direction="output" type="xmmreg" />
        <description>Return Maximum Scalar Double-Precision Floating-Point Value</descriptio
n>
    </op>
    <op bitmask="000011111010111011110000" mnemonic="MFENCE">
        <description>Memory Fence</description>
    </op>
    <op bitmask="01100110000011110101101" mnemonic="MINPD" detail="mem to xmmreg">
        <arg direction="input" type="mem" />
        <arg direction="output" type="xmmreg" />
        <description>Return Minimum Packed Double-Precision Floating-Point Values</descripti
on>
    </op>
    <op bitmask="01100110000011110101110111" mnemonic="MINPD" detail="xmmreg to xmmreg">
        <arg direction="input" type="xmmreg" />
        <arg direction="output" type="xmmreg" />
        <description>Return Minimum Packed Double-Precision Floating-Point Values</descripti
on>
    </op>
    <op bitmask="0000111101011101" mnemonic="MINPS" detail="mem to xmmreg">
        <arg direction="input" type="mem" />
        <arg direction="output" type="xmmreg" />
        <description>Return Minimum Packed Double-Precision Floating-Point Values</descripti
on>
    </op>
    <op bitmask="0000111101011110111" mnemonic="MINPS" detail="xmmreg to xmmreg">
        <arg direction="input" type="xmmreg" />
        <arg direction="output" type="xmmreg" />
        <description>Return Minimum Packed Double-Precision Floating-Point Values</descripti
on>
    </op>
    <op bitmask="11110010000011110101101" mnemonic="MINSD" detail="mem to xmmreg">
        <arg direction="input" type="mem" />
        <arg direction="output" type="xmmreg" />
        <description>Return Minimum Scalar Double-Precision Floating-Point Value</descriptio
n>
    </op>
    <op bitmask="11110010000011110101110111" mnemonic="MINSD" detail="xmmreg to xmmreg">
        <arg direction="input" type="xmmreg" />
        <arg direction="output" type="xmmreg" />
        <description>Return Minimum Scalar Double-Precision Floating-Point Value</descriptio
n>
    </op>
    <op bitmask="11110011000011110101101" mnemonic="MINSS" detail="mem to xmmreg">
        <arg direction="input" type="mem" />
        <arg direction="output" type="xmmreg" />
        <description>Return Minimum Scalar Double-Precision Floating-Point Value</descriptio
n>
    </op>
```

```
    <op bitmask="11110011000011110101110111" mnemonic="MINSS" detail="xmmreg to xmmreg">
        <arg direction="input" type="xmmreg" />
        <arg direction="output" type="xmmreg" />
        <description>Return Minimum Scalar Double-Precision Floating-Point Value</descriptio
n>
    </op>
    <op bitmask="00001111000000001111001000" mnemonic="MONITOR" detail="eax, ecx, edx">
        <arg direction="input" type="reg" />
        <arg direction="input" type="reg" />
        <arg direction="input" type="reg" />
        <description>Set up a linear address range to be monitored by hardware</description>
    </op>
    <op bitmask="000011110010000011" mnemonic="MOV" detail="register from CR0-CR4">
        <arg direction="input" type="CRx" />
        <arg direction="output" type="reg" />
        <description>Move to/from Control Registers</description>
    </op>
    <op bitmask="000011110010000111" mnemonic="MOV" detail="register from DR6-DR7, register
from DR4-DR5, register from DR0-DR3">
        <arg direction="input" type="DRx" />
        <arg direction="output" type="reg" />
        <description>Move to/from Debug Registers</description>
    </op>
    <op bitmask="000011110010001011000" mnemonic="MOV" detail="CR0 from register">
        <arg direction="input" type="reg" />
        <arg direction="output" type="CRx" />
        <description>Move to/from Control Registers</description>
    </op>
    <op bitmask="000011110010001011010" mnemonic="MOV" detail="CR2 from register">
        <arg direction="input" type="reg" />
        <arg direction="output" type="CRx" />
        <description>Move to/from Control Registers</description>
    </op>
    <op bitmask="000011110010001011011" mnemonic="MOV" detail="CR3 from register">
        <arg direction="input" type="reg" />
        <arg direction="output" type="CRx" />
        <description>Move to/from Control Registers</description>
    </op>
    <op bitmask="000011110010001011100" mnemonic="MOV" detail="CR4 from register">
        <arg direction="input" type="reg" />
        <arg direction="output" type="CRx" />
        <description>Move to/from Control Registers</description>
    </op>
    <op bitmask="000011110010001111" mnemonic="MOV" detail="DR0-DR3 from register, DR4-DR5 f
rom register, DR6-DR7 from register">
        <arg direction="input" type="reg" />
        <arg direction="output" type="DRx" />
        <description>Move to/from Debug Registers</description>
    </op>
    <op bitmask="1000100" mnemonic="MOV" detail="reg to memory">
        <arg direction="input" type="reg" />
        <arg direction="output" type="mem" />
        <description>Move Data</description>
    </op>
    <op bitmask="1000100x11" mnemonic="MOV" detail="register1 to register2">
        <arg direction="input" type="reg" />
        <arg direction="output" type="reg" />
        <description>Move Data</description>
    </op>
    <op bitmask="1000101" mnemonic="MOV" detail="memory to reg">
        <arg direction="input" type="mem" />
        <arg direction="output" type="reg" />
        <description>Move Data</description>
    </op>
    <op bitmask="1000101x11" mnemonic="MOV" detail="register2 to register1">
```

```
        <arg direction="input" type="reg" />
        <arg direction="output" type="reg" />
        <description>Move Data</description>
    </op>
    <op bitmask="10001100" mnemonic="MOV" detail="segment register to memory">
        <arg direction="input" type="SRx" />
        <arg direction="output" type="mem" />
        <description>Move to/from Segment Registers</description>
    </op>
    <op bitmask="1000110011" mnemonic="MOV" detail="segment register to register">
        <arg direction="input" type="SRx" />
        <arg direction="output" type="reg" />
        <description>Move to/from Segment Registers</description>
    </op>
    <op bitmask="10001110" mnemonic="MOV" detail="memory to segment reg, memory to SS">
        <arg direction="input" type="mem" />
        <arg direction="output" type="SRx" />
        <description>Move to/from Segment Registers</description>
    </op>
    <op bitmask="1000111011" mnemonic="MOV" detail="register to segment register, register t
o SS">
        <arg direction="input" type="reg" />
        <arg direction="output" type="SRx" />
        <description>Move to/from Segment Registers</description>
    </op>
    <op bitmask="1010000" mnemonic="MOV" detail="memory to AL, AX, or EAX">
        <arg direction="input" type="mem" />
        <arg direction="output" type="reg" />
        <description>Move Data</description>
    </op>
    <op bitmask="1010001" mnemonic="MOV" detail="AL, AX, or EAX to memory">
        <arg direction="input" type="reg" />
        <arg direction="output" type="mem" />
        <description>Move Data</description>
    </op>
    <op bitmask="1011" mnemonic="MOV" detail="immediate to register (alternate encoding)">
        <arg direction="input" type="imm" />
        <arg direction="output" type="reg" />
        <description>Move Data</description>
    </op>
    <op bitmask="1100011x11000" mnemonic="MOV" detail="immediate to register">
        <arg direction="input" type="imm" />
        <arg direction="output" type="reg" />
        <description>Move Data</description>
    </op>
    <op bitmask="1100011xxx000" mnemonic="MOV" detail="immediate to memory">
        <arg direction="input" type="imm" />
        <arg direction="output" type="mem" />
        <description>Move Data</description>
    </op>
    <op bitmask="011001100000111100101000" mnemonic="MOVAPD" detail="xmmreg1 to mem">
        <arg direction="input" type="xmmreg" />
        <arg direction="output" type="mem" />
        <description>Move Aligned Packed Double-Precision Floating-Point Values</description
>
    </op>
    <op bitmask="0110011000001111010100011" mnemonic="MOVAPD" detail="xmmreg1 to xmmreg2">
        <arg direction="input" type="xmmreg" />
        <arg direction="output" type="xmmreg" />
        <description>Move Aligned Packed Double-Precision Floating-Point Values</description
>
    </op>
    <op bitmask="011001100000111100101001" mnemonic="MOVAPD" detail="mem to xmmreg1">
        <arg direction="input" type="mem" />
        <arg direction="output" type="xmmreg" />
```

```
        <description>Move Aligned Packed Double-Precision Floating-Point Values</description
>
    </op>
    <op bitmask="01100110000011110010100111" mnemonic="MOVAPD" detail="xmmreg2 to xmmreg1">
        <arg direction="input" type="xmmreg" />
        <arg direction="output" type="xmmreg" />
        <description>Move Aligned Packed Double-Precision Floating-Point Values</description
>
    </op>
    <op bitmask="0000111100101000" mnemonic="MOVAPS" detail="mem to xmmreg1">
        <arg direction="input" type="mem" />
        <arg direction="output" type="xmmreg" />
        <description>Move Aligned Packed Single-Precision Floating-Point Values</description
>
    </op>
    <op bitmask="00001111010100011" mnemonic="MOVAPS" detail="xmmreg2 to xmmreg1">
        <arg direction="input" type="xmmreg" />
        <arg direction="output" type="xmmreg" />
        <description>Move Aligned Packed Single-Precision Floating-Point Values</description
>
    </op>
    <op bitmask="0000111100101001" mnemonic="MOVAPS" detail="xmmreg1 to mem">
        <arg direction="input" type="xmmreg" />
        <arg direction="output" type="mem" />
        <description>Move Aligned Packed Single-Precision Floating-Point Values</description
>
    </op>
    <op bitmask="00001111010100111" mnemonic="MOVAPS" detail="xmmreg1 to xmmreg2">
        <arg direction="input" type="xmmreg" />
        <arg direction="output" type="xmmreg" />
        <description>Move Aligned Packed Single-Precision Floating-Point Values</description
>
    </op>
    <op bitmask="0000111101101110" mnemonic="MOVD" detail="mem to mmxreg">
        <arg direction="input" type="mem" />
        <arg direction="output" type="mmxreg" />
        <description>Move doubleword</description>
    </op>
    <op bitmask="00001111010110111011" mnemonic="MOVD" detail="reg to mmreg">
        <arg direction="input" type="reg" />
        <arg direction="output" type="mmreg" />
        <description>Move doubleword</description>
    </op>
    <op bitmask="0000111101111110" mnemonic="MOVD" detail="mem from mmxreg">
        <arg direction="input" type="mmxreg" />
        <arg direction="output" type="mem" />
        <description>Move doubleword</description>
    </op>
    <op bitmask="00001111010111111011" mnemonic="MOVD" detail="reg from mmxreg">
        <arg direction="input" type="mmxreg" />
        <arg direction="output" type="reg" />
        <description>Move doubleword</description>
    </op>
    <op bitmask="011001100000111101101110" mnemonic="MOVD" detail="mem to xmmreg">
        <arg direction="input" type="mem" />
        <arg direction="output" type="xmmreg" />
        <description>Move Doubleword</description>
    </op>
    <op bitmask="01100110000011110110111011" mnemonic="MOVD" detail="reg to xmmeg">
        <arg direction="input" type="reg" />
        <arg direction="output" type="xmmeg" />
        <description>Move Doubleword</description>
    </op>
    <op bitmask="011001100000111101111110" mnemonic="MOVD" detail="mem from xmmreg">
        <arg direction="input" type="xmmreg" />
```

```
        <arg direction="output" type="mem" />
        <description>Move Doubleword</description>
    </op>
    <op bitmask="01100110000011110111111011" mnemonic="MOVD" detail="reg from xmmreg">
        <arg direction="input" type="xmmreg" />
        <arg direction="output" type="reg" />
        <description>Move Doubleword</description>
    </op>
    <op bitmask="11110010000011110010010" mnemonic="MOVDDUP" detail="mem to xmmreg">
        <arg direction="input" type="mem" />
        <arg direction="output" type="xmmreg" />
        <description>Move 64 bits representing one DP data from XMM2/Mem to XMM1 and duplica
te</description>
    </op>
    <op bitmask="11110010000011110001001011" mnemonic="MOVDDUP" detail="xmmreg2 to xmmreg1">
        <arg direction="input" type="xmmreg" />
        <arg direction="output" type="xmmreg" />
        <description>Move 64 bits representing one DP data from XMM2/Mem to XMM1 and duplica
te</description>
    </op>
    <op bitmask="11110010000011111101011011" mnemonic="MOVDQ2Q" detail="xmmreg to mmreg">
        <arg direction="input" type="xmmreg" />
        <arg direction="output" type="mmreg" />
        <description>Move Quadword from XMM to MMX Register</description>
    </op>
    <op bitmask="01100110000011101101111" mnemonic="MOVDQA" detail="mem to xmmreg">
        <arg direction="input" type="mem" />
        <arg direction="output" type="xmmreg" />
        <description>Move Aligned Double Quadword</description>
    </op>
    <op bitmask="01100110000011110110111111" mnemonic="MOVDQA" detail="xmmreg to xmmreg">
        <arg direction="input" type="xmmreg" />
        <arg direction="output" type="xmmreg" />
        <description>Move Aligned Double Quadword</description>
    </op>
    <op bitmask="01100110000011101111111" mnemonic="MOVDQA" detail="mem from xmmreg">
        <arg direction="input" type="xmmreg" />
        <arg direction="output" type="mem" />
        <description>Move Aligned Double Quadword</description>
    </op>
    <op bitmask="01100110000011110111111111" mnemonic="MOVDQA" detail="xmmreg to xmmreg">
        <arg direction="input" type="xmmreg" />
        <arg direction="output" type="xmmreg" />
        <description>Move Aligned Double Quadword</description>
    </op>
    <op bitmask="11110011000011101101111" mnemonic="MOVDQU" detail="mem to xmmreg">
        <arg direction="input" type="mem" />
        <arg direction="output" type="xmmreg" />
        <description>Move Unaligned Double Quadword</description>
    </op>
    <op bitmask="11110011000011110110111111" mnemonic="MOVDQU" detail="xmmreg to xmmreg">
        <arg direction="input" type="xmmreg" />
        <arg direction="output" type="xmmreg" />
        <description>Move Unaligned Double Quadword</description>
    </op>
    <op bitmask="11110011000011101111111" mnemonic="MOVDQU" detail="mem from xmmreg">
        <arg direction="input" type="xmmreg" />
        <arg direction="output" type="mem" />
        <description>Move Unaligned Double Quadword</description>
    </op>
    <op bitmask="11110011000011110111111111" mnemonic="MOVDQU" detail="xmmreg to xmmreg">
        <arg direction="input" type="xmmreg" />
        <arg direction="output" type="xmmreg" />
        <description>Move Unaligned Double Quadword</description>
    </op>
```

```
    <op bitmask="0000111100001001011" mnemonic="MOVHLPS" detail="xmmreg to xmmreg">
        <arg direction="input" type="xmmreg" />
        <arg direction="output" type="xmmreg" />
        <description>Move Packed Single-Precision Floating-Point Values High to Low</descrip
tion>
    </op>
    <op bitmask="011001100000111100010110" mnemonic="MOVHPD" detail="xmmreg to mem">
        <arg direction="input" type="xmmreg" />
        <arg direction="output" type="mem" />
        <description>Move High Packed Double-Precision Floating-Point Values</description>
    </op>
    <op bitmask="011001100000111100010111" mnemonic="MOVHPD" detail="mem to xmmreg">
        <arg direction="input" type="mem" />
        <arg direction="output" type="xmmreg" />
        <description>Move High Packed Double-Precision Floating-Point Values</description>
    </op>
    <op bitmask="0000111100010110" mnemonic="MOVHPS" detail="mem to xmmreg">
        <arg direction="input" type="mem" />
        <arg direction="output" type="xmmreg" />
        <description>Move High Packed Single-Precision Floating-Point Values</description>
    </op>
    <op bitmask="0000111100010111" mnemonic="MOVHPS" detail="xmmreg to mem">
        <arg direction="input" type="xmmreg" />
        <arg direction="output" type="mem" />
        <description>Move High Packed Single-Precision Floating-Point Values</description>
    </op>
    <op bitmask="0000111100001011011" mnemonic="MOVLHPS" detail="xmmreg to xmmreg">
        <arg direction="input" type="xmmreg" />
        <arg direction="output" type="xmmreg" />
        <description>Move Packed Single-Precision Floating-Point Values Low to High</descrip
tion>
    </op>
    <op bitmask="011001100000111100010010" mnemonic="MOVLPD" detail="xmmreg to mem">
        <arg direction="input" type="xmmreg" />
        <arg direction="output" type="mem" />
        <description>Move Low Packed Double-Precision Floating-Point Values</description>
    </op>
    <op bitmask="011001100000111100010011" mnemonic="MOVLPD" detail="mem to xmmreg">
        <arg direction="input" type="mem" />
        <arg direction="output" type="xmmreg" />
        <description>Move Low Packed Double-Precision Floating-Point Values</description>
    </op>
    <op bitmask="0000111100010010" mnemonic="MOVLPS" detail="mem to xmmreg">
        <arg direction="input" type="mem" />
        <arg direction="output" type="xmmreg" />
        <description>Move Low Packed Single-Precision Floating-Point Values</description>
    </op>
    <op bitmask="0000111100010011" mnemonic="MOVLPS" detail="xmmreg to mem">
        <arg direction="input" type="xmmreg" />
        <arg direction="output" type="mem" />
        <description>Move Low Packed Single-Precision Floating-Point Values</description>
    </op>
    <op bitmask="011001100000111110101000011" mnemonic="MOVMSKPD" detail="xmmreg to r32">
        <arg direction="input" type="xmmreg" />
        <arg direction="output" type="reg" />
        <description>Extract Packed Double-Precision Floating-Point Sign Mask</description>
    </op>
    <op bitmask="0000111110101000011" mnemonic="MOVMSKPS" detail="xmmreg to r32">
        <arg direction="input" type="xmmreg" />
        <arg direction="output" type="reg" />
        <description>Extract Packed Single-Precision Floating-Point Sign Mask</description>
    </op>
    <op bitmask="011001100000111111100111" mnemonic="MOVNTDQ" detail="xmmreg to mem">
        <arg direction="input" type="xmmreg" />
        <arg direction="output" type="mem" />
```

```
        <description>Store Double Quadword Using Non-Temporal Hint</description>
    </op>
    <op bitmask="0000111111000011" mnemonic="MOVNTI" detail="reg to mem">
        <arg direction="input" type="reg" />
        <arg direction="output" type="mem" />
        <description>Store Doubleword Using Non-Temporal Hint</description>
    </op>
    <op bitmask="011001100000111100101011" mnemonic="MOVNTPD" detail="xmmreg to mem">
        <arg direction="input" type="xmmreg" />
        <arg direction="output" type="mem" />
        <description>Store Packed Double-Precision Floating-Point Values Using Non-Temporal
Hint</description>
    </op>
    <op bitmask="0000111100101011" mnemonic="MOVNTPS" detail="xmmreg to mem">
        <arg direction="input" type="xmmreg" />
        <arg direction="output" type="mem" />
        <description>Store Packed Single-Precision Floating-Point Values Using Non-Temporal
Hint</description>
    </op>
    <op bitmask="0000111111100111" mnemonic="MOVNTQ" detail="mmreg to mem">
        <arg direction="input" type="mmreg" />
        <arg direction="output" type="mem" />
        <description>Store Quadword Using Non-Temporal Hint</description>
    </op>
    <op bitmask="0000111101101111" mnemonic="MOVQ" detail="mem to mmxreg">
        <arg direction="input" type="mem" />
        <arg direction="output" type="mmxreg" />
        <description>Move quadword</description>
    </op>
    <op bitmask="0000111110110111111" mnemonic="MOVQ" detail="mmxreg2 to mmxreg1">
        <arg direction="input" type="mmxreg" />
        <arg direction="output" type="mmxreg" />
        <description>Move quadword</description>
    </op>
    <op bitmask="0000111101111111" mnemonic="MOVQ" detail="mem from mmxreg">
        <arg direction="input" type="mmxreg" />
        <arg direction="output" type="mem" />
        <description>Move quadword</description>
    </op>
    <op bitmask="0000111110111111111" mnemonic="MOVQ" detail="mmxreg2 from mmxreg1">
        <arg direction="input" type="mmxreg" />
        <arg direction="output" type="mmxreg" />
        <description>Move quadword</description>
    </op>
    <op bitmask="011001100000111111010110" mnemonic="MOVQ" detail="mem from xmmreg">
        <arg direction="input" type="xmmreg" />
        <arg direction="output" type="mem" />
        <description>Move Quadword</description>
    </op>
    <op bitmask="011001100000111111101011011" mnemonic="MOVQ" detail="xmmreg2 from xmmreg1">
        <arg direction="input" type="xmmreg" />
        <arg direction="output" type="xmmreg" />
        <description>Move Quadword</description>
    </op>
    <op bitmask="111100110000111101111110" mnemonic="MOVQ" detail="mem to xmmreg">
        <arg direction="input" type="mem" />
        <arg direction="output" type="xmmreg" />
        <description>Move Quadword</description>
    </op>
    <op bitmask="111100110000111110111111011" mnemonic="MOVQ" detail="xmmreg2 to xmmreg1">
        <arg direction="input" type="xmmreg" />
        <arg direction="output" type="xmmreg" />
        <description>Move Quadword</description>
    </op>
    <op bitmask="111100110000111111101011011" mnemonic="MOVQ2DQ" detail="mmreg to xmmreg">
```

```
        <arg direction="input" type="mmreg" />
        <arg direction="output" type="xmmreg" />
        <description>Move Quadword from MMX to XMM Register</description>
    </op>
    <op bitmask="1010010" mnemonic="MOVS/MOVSB/MOVSW/MOVSD">
        <description>Move Data from String to String</description>
    </op>
    <op bitmask="111100100000111100010000" mnemonic="MOVSD" detail="xmmreg1 to mem">
        <arg direction="input" type="xmmreg" />
        <arg direction="output" type="mem" />
        <description>Move Scalar Double-Precision Floating-Point Values</description>
    </op>
    <op bitmask="111100100000111110001000011" mnemonic="MOVSD" detail="xmmreg1 to xmmreg2">
        <arg direction="input" type="xmmreg" />
        <arg direction="output" type="xmmreg" />
        <description>Move Scalar Double-Precision Floating-Point Values</description>
    </op>
    <op bitmask="111100100000111100010001" mnemonic="MOVSD" detail="mem to xmmreg1">
        <arg direction="input" type="mem" />
        <arg direction="output" type="xmmreg" />
        <description>Move Scalar Double-Precision Floating-Point Values</description>
    </op>
    <op bitmask="111100100000111110001000111" mnemonic="MOVSD" detail="xmmreg2 to xmmreg1">
        <arg direction="input" type="xmmreg" />
        <arg direction="output" type="xmmreg" />
        <description>Move Scalar Double-Precision Floating-Point Values</description>
    </op>
    <op bitmask="111100110000111100010110" mnemonic="MOVSHDUP" detail="mem to xmmreg">
        <arg direction="input" type="mem" />
        <arg direction="output" type="xmmreg" />
        <description>Move 128 bits representing 4 SP data from XMM2/Mem to XMM1 and duplicat
e high</description>
    </op>
    <op bitmask="111100110000111110001011011" mnemonic="MOVSHDUP" detail="xmmreg2 to xmmreg1"
>
        <arg direction="input" type="xmmreg" />
        <arg direction="output" type="xmmreg" />
        <description>Move 128 bits representing 4 SP data from XMM2/Mem to XMM1 and duplicat
e high</description>
    </op>
    <op bitmask="111100110000111100010010" mnemonic="MOVSLDUP" detail="mem to xmmreg">
        <arg direction="input" type="mem" />
        <arg direction="output" type="xmmreg" />
        <description>Move 128 bits representing 4 SP data from XMM2/Mem to XMM1 and duplicat
e low 0 - Destination is ST(0) 1 - Destination is ST(i)</description>
    </op>
    <op bitmask="111100110000111110001001011" mnemonic="MOVSLDUP" detail="xmmreg2 to xmmreg1"
>
        <arg direction="input" type="xmmreg" />
        <arg direction="output" type="xmmreg" />
        <description>Move 128 bits representing 4 SP data from XMM2/Mem to XMM1 and duplicat
e low</description>
    </op>
    <op bitmask="111100110000111100010000" mnemonic="MOVSS" detail="mem to xmmreg1">
        <arg direction="input" type="mem" />
        <arg direction="output" type="xmmreg" />
        <description>Move Scalar Single-Precision Floating-Point Values</description>
    </op>
    <op bitmask="111100110000111110001000011" mnemonic="MOVSS" detail="xmmreg2 to xmmreg1">
        <arg direction="input" type="xmmreg" />
        <arg direction="output" type="xmmreg" />
        <description>Move Scalar Single-Precision Floating-Point Values</description>
    </op>
    <op bitmask="111100110000111100010001" mnemonic="MOVSS" detail="xmmreg1 to mem">
        <arg direction="input" type="xmmreg" />
```

```
        <arg direction="output" type="mem" />
        <description>Move Scalar Single-Precision Floating-Point Values</description>
    </op>
    <op bitmask="11110011000011110001000111" mnemonic="MOVSS" detail="xmmreg1 to xmmreg2">
        <arg direction="input" type="xmmreg" />
        <arg direction="output" type="xmmreg" />
        <description>Move Scalar Single-Precision Floating-Point Values</description>
    </op>
    <op bitmask="000011111011111" mnemonic="MOVSX" detail="memory to reg">
        <arg direction="input" type="mem" />
        <arg direction="output" type="reg" />
        <description>Move with Sign-Extend</description>
    </op>
    <op bitmask="000011111011111x11" mnemonic="MOVSX" detail="register2 to register1">
        <arg direction="input" type="reg" />
        <arg direction="output" type="reg" />
        <description>Move with Sign-Extend</description>
    </op>
    <op bitmask="011001100000111100010000" mnemonic="MOVUPD" detail="xmmreg1 to mem">
        <arg direction="input" type="xmmreg" />
        <arg direction="output" type="mem" />
        <description>Move Unaligned Packed Double-Precision Floating-Point Values</descripti
on>
    </op>
    <op bitmask="011001100000111110001000011" mnemonic="MOVUPD" detail="xmmreg1 to xmmreg2">
        <arg direction="input" type="xmmreg" />
        <arg direction="output" type="xmmreg" />
        <description>Move Unaligned Packed Double-Precision Floating-Point Values</descripti
on>
    </op>
    <op bitmask="011001100000111100010001" mnemonic="MOVUPD" detail="mem to xmmreg1">
        <arg direction="input" type="mem" />
        <arg direction="output" type="xmmreg" />
        <description>Move Unaligned Packed Double-Precision Floating-Point Values</descripti
on>
    </op>
    <op bitmask="011001100000111110001000111" mnemonic="MOVUPD" detail="xmmreg2 to xmmreg1">
        <arg direction="input" type="xmmreg" />
        <arg direction="output" type="xmmreg" />
        <description>Move Unaligned Packed Double-Precision Floating-Point Values</descripti
on>
    </op>
    <op bitmask="0000111100010000" mnemonic="MOVUPS" detail="mem to xmmreg1">
        <arg direction="input" type="mem" />
        <arg direction="output" type="xmmreg" />
        <description>Move Unaligned Packed Single-Precision Floating-Point Values</descripti
on>
    </op>
    <op bitmask="000011110001000011" mnemonic="MOVUPS" detail="xmmreg2 to xmmreg1">
        <arg direction="input" type="xmmreg" />
        <arg direction="output" type="xmmreg" />
        <description>Move Unaligned Packed Single-Precision Floating-Point Values</descripti
on>
    </op>
    <op bitmask="0000111100010001" mnemonic="MOVUPS" detail="xmmreg1 to mem">
        <arg direction="input" type="xmmreg" />
        <arg direction="output" type="mem" />
        <description>Move Unaligned Packed Single-Precision Floating-Point Values</descripti
on>
    </op>
    <op bitmask="000011110001000111" mnemonic="MOVUPS" detail="xmmreg1 to xmmreg2">
        <arg direction="input" type="xmmreg" />
        <arg direction="output" type="xmmreg" />
        <description>Move Unaligned Packed Single-Precision Floating-Point Values</descripti
on>
```

```
    </op>
    <op bitmask="000011111011011" mnemonic="MOVZX" detail="memory to register">
        <arg direction="input" type="mem" />
        <arg direction="output" type="reg" />
        <description>Move with Zero-Extend</description>
    </op>
    <op bitmask="000011111011011x11" mnemonic="MOVZX" detail="register2 to register1">
        <arg direction="input" type="reg" />
        <arg direction="output" type="reg" />
        <description>Move with Zero-Extend</description>
    </op>
    <op bitmask="1111011x11100" mnemonic="MUL" detail="AL, AX, or EAX with register">
        <arg direction="input" type="reg" />
        <arg direction="input" type="reg" />
        <arg direction="output" type="reg" />
        <description>Unsigned Multiply</description>
    </op>
    <op bitmask="1111011xxx100" mnemonic="MUL" detail="AL, AX, or EAX with memory">
        <arg direction="input" type="reg" />
        <arg direction="input" type="mem" />
        <arg direction="output" type="reg" />
        <description>Unsigned Multiply</description>
    </op>
    <op bitmask="011001100000111101011001" mnemonic="MULPD" detail="mem to xmmreg">
        <arg direction="input" type="mem" />
        <arg direction="output" type="xmmreg" />
        <description>Multiply Packed Double-Precision Floating-Point Values</description>
    </op>
    <op bitmask="011001100000111110101100111" mnemonic="MULPD" detail="xmmreg to xmmreg">
        <arg direction="input" type="xmmreg" />
        <arg direction="output" type="xmmreg" />
        <description>Multiply Packed Double-Precision Floating-Point Values</description>
    </op>
    <op bitmask="0000111101011001" mnemonic="MULPS" detail="mem to xmmreg">
        <arg direction="input" type="mem" />
        <arg direction="output" type="xmmreg" />
        <description>Multiply Packed Single-Precision Floating-Point Values</description>
    </op>
    <op bitmask="000011110101100111" mnemonic="MULPS" detail="xmmreg to xmmreg">
        <arg direction="input" type="xmmreg" />
        <arg direction="output" type="xmmreg" />
        <description>Multiply Packed Single-Precision Floating-Point Values</description>
    </op>
    <op bitmask="111100100000111101011001" mnemonic="MULSD" detail="mem to xmmreg">
        <arg direction="input" type="mem" />
        <arg direction="output" type="xmmreg" />
        <description>Multiply Scalar Double-Precision Floating-Point Values</description>
    </op>
    <op bitmask="111100100000111110101100111" mnemonic="MULSD" detail="xmmreg to xmmreg">
        <arg direction="input" type="xmmreg" />
        <arg direction="output" type="xmmreg" />
        <description>Multiply Scalar Double-Precision Floating-Point Values</description>
    </op>
    <op bitmask="111100110000111101011001" mnemonic="MULSS" detail="mem to xmmreg">
        <arg direction="input" type="mem" />
        <arg direction="output" type="xmmreg" />
        <description>Multiply Scalar Single-Precision Floating-Point Values</description>
    </op>
    <op bitmask="111100110000111110101100111" mnemonic="MULSS" detail="xmmreg to xmmreg">
        <arg direction="input" type="xmmreg" />
        <arg direction="output" type="xmmreg" />
        <description>Multiply Scalar Single-Precision Floating-Point Values</description>
    </op>
    <op bitmask="000011110000000111001001" mnemonic="MWAIT" detail="eax, ecx">
        <arg direction="input" type="reg" />
```

```
        <arg direction="input" type="reg" />
        <description>Wait until write-back store performed within the range specified by the
 instruction MONITOR</description>
    </op>
    <op bitmask="1111011x11011" mnemonic="NEG" detail="register">
        <arg direction="input" type="reg" />
        <arg direction="output" type="reg" />
        <description>Two’s Complement Negation</description>
    </op>
    <op bitmask="1111011xxx011" mnemonic="NEG" detail="memory">
        <arg direction="input" type="mem" />
        <arg direction="output" type="mem" />
        <description>Two’s Complement Negation</description>
    </op>
    <op bitmask="10010000" mnemonic="NOP">
        <description>No Operation</description>
    </op>
    <op bitmask="1111011x11010" mnemonic="NOT" detail="register">
        <arg direction="input" type="reg" />
        <arg direction="output" type="reg" />
        <description>One’s Complement Negation</description>
    </op>
    <op bitmask="1111011xxx010" mnemonic="NOT" detail="memory">
        <arg direction="input" type="mem" />
        <arg direction="output" type="mem" />
        <description>One’s Complement Negation</description>
    </op>
    <op bitmask="0000100" mnemonic="OR" detail="register to memory">
        <arg direction="input" type="mem" />
        <arg direction="input" type="reg" />
        <arg direction="output" type="mem" />
        <description>Logical Inclusive OR</description>
    </op>
    <op bitmask="0000100x11" mnemonic="OR" detail="register1 to register2">
        <arg direction="input" type="reg" />
        <arg direction="input" type="reg" />
        <arg direction="output" type="reg" />
        <description>Logical Inclusive OR</description>
    </op>
    <op bitmask="0000101" mnemonic="OR" detail="memory to register">
        <arg direction="input" type="reg" />
        <arg direction="input" type="mem" />
        <arg direction="output" type="reg" />
        <description>Logical Inclusive OR</description>
    </op>
    <op bitmask="0000101x11" mnemonic="OR" detail="register2 to register1">
        <arg direction="input" type="reg" />
        <arg direction="input" type="reg" />
        <arg direction="output" type="reg" />
        <description>Logical Inclusive OR</description>
    </op>
    <op bitmask="0000110" mnemonic="OR" detail="immediate to AL, AX, or EAX">
        <arg direction="input" type="reg" />
        <arg direction="input" type="imm" />
        <arg direction="output" type="reg" />
        <description>Logical Inclusive OR</description>
    </op>
    <op bitmask="100000xx11001" mnemonic="OR" detail="immediate to register">
        <arg direction="input" type="reg" />
        <arg direction="input" type="imm" />
        <arg direction="output" type="reg" />
        <description>Logical Inclusive OR</description>
    </op>
    <op bitmask="100000xxxx001" mnemonic="OR" detail="immediate to memory">
        <arg direction="input" type="mem" />
```

```
        <arg direction="input" type="imm" />
        <arg direction="output" type="mem" />
        <description>Logical Inclusive OR</description>
    </op>
    <op bitmask="01100110000011110101010110" mnemonic="ORPD" detail="mem to xmmreg">
        <arg direction="input" type="xmmreg" />
        <arg direction="input" type="mem" />
        <arg direction="output" type="xmmreg" />
        <description>Bitwise Logical OR of Double-Precision Floating-Point Values</descripti
on>
    </op>
    <op bitmask="011001100000111110101011011" mnemonic="ORPD" detail="xmmreg to xmmreg">
        <arg direction="input" type="xmmreg" />
        <arg direction="input" type="xmmreg" />
        <arg direction="output" type="xmmreg" />
        <description>Bitwise Logical OR of Double-Precision Floating-Point Values</descripti
on>
    </op>
    <op bitmask="0000111101010110" mnemonic="ORPS" detail="mem to xmmreg">
        <arg direction="input" type="xmmreg" />
        <arg direction="input" type="mem" />
        <arg direction="output" type="xmmreg" />
        <description>Bitwise Logical OR of Single-Precision Floating-Point Values</descripti
on>
    </op>
    <op bitmask="000011110101011011" mnemonic="ORPS" detail="xmmreg to xmmreg">
        <arg direction="input" type="xmmreg" />
        <arg direction="input" type="xmmreg" />
        <arg direction="output" type="xmmreg" />
        <description>Bitwise Logical OR of Single-Precision Floating-Point Values</descripti
on>
    </op>
    <op bitmask="1110011" mnemonic="OUT" detail="fixed port">
        <arg direction="input" type="reg" />
        <arg direction="output" type="imm" />
        <description>Output to Port</description>
    </op>
    <op bitmask="1110111" mnemonic="OUT" detail="variable port">
        <arg direction="input" type="reg" />
        <arg direction="output" type="imm" />
        <description>Output to Port</description>
    </op>
    <op bitmask="0110111" mnemonic="OUTS">
        <description>Output to DX Port</description>
    </op>
    <op bitmask="0000111101101011" mnemonic="PACKSSDW" detail="memory to mmxreg">
        <arg direction="input" type="mem" />
        <arg direction="output" type="mmxreg" />
        <description>Pack dword to word data (signed with saturation)</description>
    </op>
    <op bitmask="000011110110101111" mnemonic="PACKSSDW" detail="mmxreg2 to mmxreg1">
        <arg direction="input" type="mmxreg" />
        <arg direction="output" type="mmxreg" />
        <description>Pack dword to word data (signed with saturation)</description>
    </op>
    <op bitmask="01100110000011110110101011" mnemonic="PACKSSDW" detail="memory to xmmreg">
        <arg direction="input" type="mem" />
        <arg direction="output" type="xmmreg" />
        <description>Pack Dword To Word Data (signed with saturation)</description>
    </op>
    <op bitmask="011001100000111110110101111" mnemonic="PACKSSDW" detail="xmmreg2 to xmmreg1"
>
        <arg direction="input" type="xmmreg" />
        <arg direction="output" type="xmmreg" />
        <description>Pack Dword To Word Data (signed with saturation)</description>
```

```
    </op>
    <op bitmask="01100110000011111011000011" mnemonic="PACKSSWB" detail="memory to xmmreg">
        <arg direction="input" type="mem" />
        <arg direction="output" type="xmmreg" />
        <description>Pack Word To Byte Data (signed with saturation)</description>
    </op>
    <op bitmask="01100110000011111011000111" mnemonic="PACKSSWB" detail="xmmreg2 to xmmreg1"
>
        <arg direction="input" type="xmmreg" />
        <arg direction="output" type="xmmreg" />
        <description>Pack Word To Byte Data (signed with saturation)</description>
    </op>
    <op bitmask="0000111101100011" mnemonic="PACKSSWB1" detail="memory to mmxreg">
        <arg direction="input" type="mem" />
        <arg direction="output" type="mmxreg" />
        <description>Pack word to byte data (signed with saturation)</description>
    </op>
    <op bitmask="000011110110001111" mnemonic="PACKSSWB1" detail="mmxreg2 to mmxreg1">
        <arg direction="input" type="mmxreg" />
        <arg direction="output" type="mmxreg" />
        <description>Pack word to byte data (signed with saturation)</description>
    </op>
    <op bitmask="0000111101100111" mnemonic="PACKUSWB" detail="memory to mmxreg">
        <arg direction="input" type="mem" />
        <arg direction="output" type="mmxreg" />
        <description>Pack word to byte data (unsigned with saturation)</description>
    </op>
    <op bitmask="000011110110011111" mnemonic="PACKUSWB" detail="mmxreg2 to mmxreg1">
        <arg direction="input" type="mmxreg" />
        <arg direction="output" type="mmxreg" />
        <description>Pack word to byte data (unsigned with saturation)</description>
    </op>
    <op bitmask="01100110000011111011000111" mnemonic="PACKUSWB" detail="memory to xmmreg">
        <arg direction="input" type="mem" />
        <arg direction="output" type="xmmreg" />
        <description>Pack Word To Byte Data (unsigned with saturation)</description>
    </op>
    <op bitmask="01100110000011111011001111" mnemonic="PACKUSWB" detail="xmmreg2 to xmmreg1"
>
        <arg direction="input" type="xmmreg" />
        <arg direction="output" type="xmmreg" />
        <description>Pack Word To Byte Data (unsigned with saturation)</description>
    </op>
    <op bitmask="00001111111111" mnemonic="PADD" detail="memory to mmxreg">
        <arg direction="input" type="mem" />
        <arg direction="output" type="mmxreg" />
        <description>Add with wrap-around</description>
    </op>
    <op bitmask="00001111111111xx11" mnemonic="PADD" detail="mmxreg2 to mmxreg1">
        <arg direction="input" type="mmxreg" />
        <arg direction="output" type="mmxreg" />
        <description>Add with wrap-around</description>
    </op>
    <op bitmask="0110011000001111111111" mnemonic="PADD" detail="memory to xmmreg">
        <arg direction="input" type="mem" />
        <arg direction="output" type="xmmreg" />
        <description>Add With Wrap-around</description>
    </op>
    <op bitmask="0110011000001111111111xx11" mnemonic="PADD" detail="xmmreg2 to xmmreg1">
        <arg direction="input" type="xmmreg" />
        <arg direction="output" type="xmmreg" />
        <description>Add With Wrap-around</description>
    </op>
    <op bitmask="0000111111010100" mnemonic="PADDQ" detail="mem to mmreg">
        <arg direction="input" type="mem" />
```

```
        <arg direction="output" type="mmreg" />
        <description>Add Packed Quadword Integers</description>
    </op>
    <op bitmask="0000111111010010011" mnemonic="PADDQ" detail="mmreg to mmreg">
        <arg direction="input" type="mmreg" />
        <arg direction="output" type="mmreg" />
        <description>Add Packed Quadword Integers</description>
    </op>
    <op bitmask="01100110000111111010100" mnemonic="PADDQ" detail="mem to xmmreg">
        <arg direction="input" type="mem" />
        <arg direction="output" type="xmmreg" />
        <description>Add Packed Quadword Integers</description>
    </op>
    <op bitmask="011001100001111111010010011" mnemonic="PADDQ" detail="xmmreg to xmmreg">
        <arg direction="input" type="xmmreg" />
        <arg direction="output" type="xmmreg" />
        <description>Add Packed Quadword Integers</description>
    </op>
    <op bitmask="00001111111011" mnemonic="PADDS" detail="memory to mmxreg">
        <arg direction="input" type="mem" />
        <arg direction="output" type="mmxreg" />
        <description>Add signed with saturation</description>
    </op>
    <op bitmask="00001111111011xx11" mnemonic="PADDS" detail="mmxreg2 to mmxreg1">
        <arg direction="input" type="mmxreg" />
        <arg direction="output" type="mmxreg" />
        <description>Add signed with saturation</description>
    </op>
    <op bitmask="01100110000011111111011" mnemonic="PADDS" detail="memory to xmmreg">
        <arg direction="input" type="mem" />
        <arg direction="output" type="xmmreg" />
        <description>Add Signed With Saturation</description>
    </op>
    <op bitmask="011001100000111111111011xx11" mnemonic="PADDS" detail="xmmreg2 to xmmreg1">
        <arg direction="input" type="xmmreg" />
        <arg direction="output" type="xmmreg" />
        <description>Add Signed With Saturation</description>
    </op>
    <op bitmask="00001111110111" mnemonic="PADDUS" detail="memory to mmxreg">
        <arg direction="input" type="mem" />
        <arg direction="output" type="mmxreg" />
        <description>Add unsigned with saturation</description>
    </op>
    <op bitmask="00001111110111xx11" mnemonic="PADDUS" detail="mmxreg2 to mmxreg1">
        <arg direction="input" type="mmxreg" />
        <arg direction="output" type="mmxreg" />
        <description>Add unsigned with saturation</description>
    </op>
    <op bitmask="01100110000011111110111" mnemonic="PADDUS" detail="memory to xmmreg">
        <arg direction="input" type="mem" />
        <arg direction="output" type="xmmreg" />
        <description>Add Unsigned With Saturation</description>
    </op>
    <op bitmask="011001100000111111110111xx11" mnemonic="PADDUS" detail="xmmreg2 to xmmreg1">
        <arg direction="input" type="xmmreg" />
        <arg direction="output" type="xmmreg" />
        <description>Add Unsigned With Saturation</description>
    </op>
    <op bitmask="0000111111011011" mnemonic="PAND" detail="memory to mmxreg">
        <arg direction="input" type="mem" />
        <arg direction="output" type="mmxreg" />
        <description>Bitwise And</description>
    </op>
    <op bitmask="0000111111101101111" mnemonic="PAND" detail="mmxreg2 to mmxreg1">
        <arg direction="input" type="mmxreg" />
```

```
        <arg direction="output" type="mmxreg" />
        <description>Bitwise And</description>
    </op>
    <op bitmask="01100110000011111011011" mnemonic="PAND" detail="memory to xmmreg">
        <arg direction="input" type="mem" />
        <arg direction="output" type="xmmreg" />
        <description>Bitwise And</description>
    </op>
    <op bitmask="011001100000111111101101111" mnemonic="PAND" detail="xmmreg2 to xmmreg1">
        <arg direction="input" type="xmmreg" />
        <arg direction="output" type="xmmreg" />
        <description>Bitwise And</description>
    </op>
    <op bitmask="0000111111011111" mnemonic="PANDN" detail="memory to mmxreg">
        <arg direction="input" type="mem" />
        <arg direction="output" type="mmxreg" />
        <description>Bitwise AndNot</description>
    </op>
    <op bitmask="0000111111101111111" mnemonic="PANDN" detail="mmxreg2 to mmxreg1">
        <arg direction="input" type="mmxreg" />
        <arg direction="output" type="mmxreg" />
        <description>Bitwise AndNot</description>
    </op>
    <op bitmask="01100110000011111011111" mnemonic="PANDN" detail="memory to xmmreg">
        <arg direction="input" type="mem" />
        <arg direction="output" type="xmmreg" />
        <description>Bitwise AndNot</description>
    </op>
    <op bitmask="011001100000111111101111111" mnemonic="PANDN" detail="xmmreg2 to xmmreg1">
        <arg direction="input" type="xmmreg" />
        <arg direction="output" type="xmmreg" />
        <description>Bitwise AndNot</description>
    </op>
    <op bitmask="1111001110010000" mnemonic="PAUSE">
        <description>Spin Loop Hint</description>
    </op>
    <op bitmask="01100110000011111100000" mnemonic="PAVGB" detail="mem to xmmreg">
        <arg direction="input" type="mem" />
        <arg direction="output" type="xmmreg" />
        <description>Average Packed Integers</description>
    </op>
    <op bitmask="011001100000111111110000011" mnemonic="PAVGB" detail="xmmreg to xmmreg">
        <arg direction="input" type="xmmreg" />
        <arg direction="output" type="xmmreg" />
        <description>Average Packed Integers</description>
    </op>
    <op bitmask="0000111111100000" mnemonic="PAVGB/PAVGW" detail="mem to mmreg">
        <arg direction="input" type="mem" />
        <arg direction="output" type="mmreg" />
        <description>Average Packed Integers</description>
    </op>
    <op bitmask="0000111111110000011" mnemonic="PAVGB/PAVGW" detail="mmreg to mmreg">
        <arg direction="input" type="mmreg" />
        <arg direction="output" type="mmreg" />
        <description>Average Packed Integers</description>
    </op>
    <op bitmask="0000111111110001111" mnemonic="PAVGB/PAVGW" detail="mmreg to mmreg">
        <arg direction="input" type="mmreg" />
        <arg direction="output" type="mmreg" />
        <description>Average Packed Integers</description>
    </op>
    <op bitmask="01100110000011111100011" mnemonic="PAVGW" detail="mem to xmmreg">
        <arg direction="input" type="mem" />
        <arg direction="output" type="xmmreg" />
        <description>Average Packed Integers</description>
```

```
    </op>
    <op bitmask="01100110000011111110001111" mnemonic="PAVGW" detail="xmmreg to xmmreg">
        <arg direction="input" type="xmmreg" />
        <arg direction="output" type="xmmreg" />
        <description>Average Packed Integers</description>
    </op>
    <op bitmask="00001111011101" mnemonic="PCMPEQ" detail="mmxreg with memory">
        <arg direction="input" type="mmxreg" />
        <arg direction="input" type="mem" />
        <description>Packed compare for equality</description>
    </op>
    <op bitmask="00001111011101xx11" mnemonic="PCMPEQ" detail="mmxreg1 with mmxreg2">
        <arg direction="input" type="mmxreg" />
        <arg direction="input" type="mmxreg" />
        <description>Packed compare for equality</description>
    </op>
    <op bitmask="01100110000011111011101" mnemonic="PCMPEQ" detail="xmmreg with memory">
        <arg direction="input" type="xmmreg" />
        <arg direction="input" type="mem" />
        <description>Packed Compare For Equality</description>
    </op>
    <op bitmask="01100110000011111011101xx11" mnemonic="PCMPEQ" detail="xmmreg1 with xmmreg2"
>
        <arg direction="input" type="xmmreg" />
        <arg direction="input" type="xmmreg" />
        <description>Packed Compare For Equality</description>
    </op>
    <op bitmask="00001111011001" mnemonic="PCMPGT" detail="mmxreg with memory">
        <arg direction="input" type="mmxreg" />
        <arg direction="input" type="mem" />
        <description>Packed compare greater (signed)</description>
    </op>
    <op bitmask="00001111011001xx11" mnemonic="PCMPGT" detail="mmxreg1 with mmxreg2">
        <arg direction="input" type="mmxreg" />
        <arg direction="input" type="mmxreg" />
        <description>Packed compare greater (signed)</description>
    </op>
    <op bitmask="01100110000011111011001" mnemonic="PCMPGT" detail="xmmreg with memory">
        <arg direction="input" type="xmmreg" />
        <arg direction="input" type="mem" />
        <description>Packed Compare Greater (signed)</description>
    </op>
    <op bitmask="01100110000011111011001xx11" mnemonic="PCMPGT" detail="xmmreg1 with xmmreg2"
>
        <arg direction="input" type="xmmreg" />
        <arg direction="input" type="xmmreg" />
        <description>Packed Compare Greater (signed)</description>
    </op>
    <op bitmask="00001111111000010111" mnemonic="PEXTRW" detail="mmreg to reg32, imm8">
        <arg direction="input" type="mmreg" />
        <arg direction="input" type="imm" />
        <arg direction="output" type="reg" />
        <description>Extract Word</description>
    </op>
    <op bitmask="0000111111100011" mnemonic="PEXTRW">
        <arg direction="input" type="mmreg" />
        <arg direction="output" type="mmreg" />
        <description>Extract Word</description>
    </op>
    <op bitmask="01100110000011111100010111" mnemonic="PEXTRW" detail="xmmreg to reg32, imm8
">
        <arg direction="input" type="xmmreg" />
        <arg direction="input" type="imm" />
        <arg direction="output" type="reg" />
        <description>Extract Word</description>
```

```
    </op>
    <op bitmask="0000111111000100" mnemonic="PINSRW" detail="m16 to mmreg, imm8">
        <arg direction="input" type="mem" />
        <arg direction="input" type="imm" />
        <arg direction="output" type="mmreg" />
        <description>Insert Word</description>
    </op>
    <op bitmask="0000111111000010011" mnemonic="PINSRW" detail="reg32 to mmreg, imm8">
        <arg direction="input" type="reg" />
        <arg direction="input" type="imm" />
        <arg direction="output" type="mmreg" />
        <description>Insert Word</description>
    </op>
    <op bitmask="01100110000011111000100" mnemonic="PINSRW" detail="m16 to xmmreg, imm8">
        <arg direction="input" type="mem" />
        <arg direction="input" type="imm" />
        <arg direction="output" type="xmmreg" />
        <description>Insert Word</description>
    </op>
    <op bitmask="01100110000011111100010011" mnemonic="PINSRW" detail="reg32 to xmmreg, imm8
">
        <arg direction="input" type="reg" />
        <arg direction="input" type="imm" />
        <arg direction="output" type="xmmreg" />
        <description>Insert Word</description>
    </op>
    <op bitmask="0000111111110101" mnemonic="PMADDWD" detail="memory to mmxreg">
        <arg direction="input" type="mem" />
        <arg direction="output" type="mmxreg" />
        <description>Packed multiply add</description>
    </op>
    <op bitmask="0000111111110010111" mnemonic="PMADDWD" detail="mmxreg2 to mmxreg1">
        <arg direction="input" type="mmxreg" />
        <arg direction="output" type="mmxreg" />
        <description>Packed multiply add</description>
    </op>
    <op bitmask="01100110000011111110101" mnemonic="PMADDWD" detail="memory to xmmreg">
        <arg direction="input" type="mem" />
        <arg direction="output" type="xmmreg" />
        <description>Packed Multiply Add</description>
    </op>
    <op bitmask="01100110000011111111010111" mnemonic="PMADDWD" detail="xmmreg2 to xmmreg1">
        <arg direction="input" type="xmmreg" />
        <arg direction="output" type="xmmreg" />
        <description>Packed Multiply Add</description>
    </op>
    <op bitmask="0000111111101110" mnemonic="PMAXSW" detail="mem to mmreg">
        <arg direction="input" type="mem" />
        <arg direction="output" type="mmreg" />
        <description>Maximum of Packed Signed Word Integers</description>
    </op>
    <op bitmask="0000111111110111011" mnemonic="PMAXSW" detail="mmreg to mmreg">
        <arg direction="input" type="mmreg" />
        <arg direction="output" type="mmreg" />
        <description>Maximum of Packed Signed Word Integers</description>
    </op>
    <op bitmask="01100110000011111101110" mnemonic="PMAXSW" detail="mem to xmmreg">
        <arg direction="input" type="mem" />
        <arg direction="output" type="xmmreg" />
        <description>Maximum of Packed Signed Word Integers</description>
    </op>
    <op bitmask="01100110000011111110111011" mnemonic="PMAXSW" detail="xmmreg to xmmreg">
        <arg direction="input" type="xmmreg" />
        <arg direction="output" type="xmmreg" />
        <description>Maximum of Packed Signed Word Integers</description>
```

```
    </op>
    <op bitmask="0000111111011110" mnemonic="PMAXUB" detail="mem to mmreg">
        <arg direction="input" type="mem" />
        <arg direction="output" type="mmreg" />
        <description>Maximum of Packed Unsigned Byte Integers</description>
    </op>
    <op bitmask="0000111111101111011" mnemonic="PMAXUB" detail="mmreg to mmreg">
        <arg direction="input" type="mmreg" />
        <arg direction="output" type="mmreg" />
        <description>Maximum of Packed Unsigned Byte Integers</description>
    </op>
    <op bitmask="01100110000011111011110" mnemonic="PMAXUB" detail="mem to xmmreg">
        <arg direction="input" type="mem" />
        <arg direction="output" type="xmmreg" />
        <description>Maximum of Packed Unsigned Byte Integers</description>
    </op>
    <op bitmask="011001100000111111101111011" mnemonic="PMAXUB" detail="xmmreg to xmmreg">
        <arg direction="input" type="xmmreg" />
        <arg direction="output" type="xmmreg" />
        <description>Maximum of Packed Unsigned Byte Integers</description>
    </op>
    <op bitmask="0000111111101010" mnemonic="PMINSW" detail="mem to mmreg">
        <arg direction="input" type="mem" />
        <arg direction="output" type="mmreg" />
        <description>Minimum of Packed Signed Word Integers</description>
    </op>
    <op bitmask="0000111111110101011" mnemonic="PMINSW" detail="mmreg to mmreg">
        <arg direction="input" type="mmreg" />
        <arg direction="output" type="mmreg" />
        <description>Minimum of Packed Signed Word Integers</description>
    </op>
    <op bitmask="01100110000011111101010" mnemonic="PMINSW" detail="mem to xmmreg">
        <arg direction="input" type="mem" />
        <arg direction="output" type="xmmreg" />
        <description>Minimum of Packed Signed Word Integers</description>
    </op>
    <op bitmask="011001100000111111110101011" mnemonic="PMINSW" detail="xmmreg to xmmreg">
        <arg direction="input" type="xmmreg" />
        <arg direction="output" type="xmmreg" />
        <description>Minimum of Packed Signed Word Integers</description>
    </op>
    <op bitmask="0000111111011010" mnemonic="PMINUB" detail="mem to mmreg">
        <arg direction="input" type="mem" />
        <arg direction="output" type="mmreg" />
        <description>Minimum of Packed Unsigned Byte Integers</description>
    </op>
    <op bitmask="0000111111101101011" mnemonic="PMINUB" detail="mmreg to mmreg">
        <arg direction="input" type="mmreg" />
        <arg direction="output" type="mmreg" />
        <description>Minimum of Packed Unsigned Byte Integers</description>
    </op>
    <op bitmask="01100110000011111011010" mnemonic="PMINUB" detail="mem to xmmreg">
        <arg direction="input" type="mem" />
        <arg direction="output" type="xmmreg" />
        <description>Minimum of Packed Unsigned Byte Integers</description>
    </op>
    <op bitmask="011001100000111111101101011" mnemonic="PMINUB" detail="xmmreg to xmmreg">
        <arg direction="input" type="xmmreg" />
        <arg direction="output" type="xmmreg" />
        <description>Minimum of Packed Unsigned Byte Integers</description>
    </op>
    <op bitmask="0000111111101011111" mnemonic="PMOVMSKB" detail="mmreg to reg32">
        <arg direction="input" type="mmreg" />
        <arg direction="output" type="reg" />
        <description>Move Byte Mask To Integer</description>
```

```
    </op>
    <op bitmask="01100110000011111101011111" mnemonic="PMOVMSKB" detail="xmmreg to reg32">
        <arg direction="input" type="xmmreg" />
        <arg direction="output" type="reg" />
        <description>Move Byte Mask To Integer</description>
    </op>
    <op bitmask="0000111111100100" mnemonic="PMULHUW" detail="memory to mmxreg, mem to mmreg
">
        <arg direction="input" type="mem" />
        <arg direction="output" type="mmreg" />
        <description>Multiply Packed Unsigned Integers and Store High Result</description>
    </op>
    <op bitmask="00001111111010011" mnemonic="PMULHUW" detail="mmxreg2 to mmxreg1, mmreg to
 mmreg">
        <arg direction="input" type="mmreg" />
        <arg direction="output" type="mmreg" />
        <description>Multiply Packed Unsigned Integers and Store High Result</description>
    </op>
    <op bitmask="01100110000011111100100" mnemonic="PMULHUW" detail="memory to xmmreg">
        <arg direction="input" type="mem" />
        <arg direction="output" type="xmmreg" />
        <description>Packed multiplication, store high word (unsigned)</description>
    </op>
    <op bitmask="01100110000011111110010011" mnemonic="PMULHUW" detail="xmmreg2 to xmmreg1">
        <arg direction="input" type="xmmreg" />
        <arg direction="output" type="xmmreg" />
        <description>Packed multiplication, store high word (unsigned)</description>
    </op>
    <op bitmask="0000111111100101" mnemonic="PMULHW" detail="memory to mmxreg">
        <arg direction="input" type="mem" />
        <arg direction="output" type="mmxreg" />
        <description>Packed multiplication, store high word</description>
    </op>
    <op bitmask="00001111111010111" mnemonic="PMULHW" detail="mmxreg2 to mmxreg1">
        <arg direction="input" type="mmxreg" />
        <arg direction="output" type="mmxreg" />
        <description>Packed multiplication, store high word</description>
    </op>
    <op bitmask="01100110000011111100101" mnemonic="PMULHW" detail="memory to xmmreg">
        <arg direction="input" type="mem" />
        <arg direction="output" type="xmmreg" />
        <description>Packed Multiplication, store high word</description>
    </op>
    <op bitmask="01100110000011111110010111" mnemonic="PMULHW" detail="xmmreg2 to xmmreg1">
        <arg direction="input" type="xmmreg" />
        <arg direction="output" type="xmmreg" />
        <description>Packed Multiplication, store high word</description>
    </op>
    <op bitmask="0000111111010101" mnemonic="PMULLW" detail="memory to mmxreg">
        <arg direction="input" type="mem" />
        <arg direction="output" type="mmxreg" />
        <description>Packed multiplication, store low word</description>
    </op>
    <op bitmask="00001111111010111" mnemonic="PMULLW" detail="mmxreg2 to mmxreg1">
        <arg direction="input" type="mmxreg" />
        <arg direction="output" type="mmxreg" />
        <description>Packed multiplication, store low word</description>
    </op>
    <op bitmask="01100110000011111110101011" mnemonic="PMULLW" detail="memory to xmmreg">
        <arg direction="input" type="mem" />
        <arg direction="output" type="xmmreg" />
        <description>Packed Multiplication, store low word</description>
    </op>
    <op bitmask="01100110000011111110101111" mnemonic="PMULLW" detail="xmmreg2 to xmmreg1">
        <arg direction="input" type="xmmreg" />
```

```
        <arg direction="output" type="xmmreg" />
        <description>Packed Multiplication, store low word</description>
    </op>
    <op bitmask="0000111111110100" mnemonic="PMULUDQ" detail="mem to mmreg">
        <arg direction="input" type="mem" />
        <arg direction="output" type="mmreg" />
        <description>Multiply Packed Unsigned Doubleword Integers</description>
    </op>
    <op bitmask="0000111111110011" mnemonic="PMULUDQ" detail="mmreg to mmreg">
        <arg direction="input" type="mmreg" />
        <arg direction="output" type="mmreg" />
        <description>Multiply Packed Unsigned Doubleword Integers</description>
    </op>
    <op bitmask="01100110000111111110100" mnemonic="PMULUDQ" detail="mem to xmmreg">
        <arg direction="input" type="mem" />
        <arg direction="output" type="xmmreg" />
        <description>Multiply Packed Unsigned Doubleword Integers</description>
    </op>
    <op bitmask="01100110000111111110011" mnemonic="PMULUDQ" detail="xmmreg to xmmreg">
        <arg direction="input" type="xmmreg" />
        <arg direction="output" type="xmmreg" />
        <description>Multiply Packed Unsigned Doubleword Integers</description>
    </op>
    <op bitmask="0000111110xx001" mnemonic="POP" detail="segment register FS, GS">
        <arg direction="output" type="SRx" />
        <description>Pop a Segment Register from the Stack </description>
    </op>
    <op bitmask="000xx111" mnemonic="POP" detail="segment register DS, ES, segment register
SS">
        <arg direction="output" type="SRx" />
        <description>Pop a Segment Register from the Stack </description>
    </op>
    <op bitmask="01011" mnemonic="POP" detail="register (alternate encoding)">
        <arg direction="output" type="reg" />
        <description>Pop a Word from the Stack</description>
    </op>
    <op bitmask="1000111111000" mnemonic="POP" detail="register">
        <arg direction="output" type="reg" />
        <description>Pop a Word from the Stack</description>
    </op>
    <op bitmask="10001111xx000" mnemonic="POP" detail="memory">
        <arg direction="output" type="mem" />
        <description>Pop a Word from the Stack</description>
    </op>
    <op bitmask="01100001" mnemonic="POPA/POPAD">
        <description>Pop All General Registers</description>
    </op>
    <op bitmask="10011101" mnemonic="POPF/POPFD">
        <arg direction="output" type="FLAGS" />
        <description>Pop Stack into FLAGS or EFLAGS Register</description>
    </op>
    <op bitmask="0000111111101011" mnemonic="POR" detail="memory to mmxreg">
        <arg direction="input" type="mem" />
        <arg direction="output" type="mmxreg" />
        <description>Bitwise Or</description>
    </op>
    <op bitmask="0000111111110101111" mnemonic="POR" detail="mmxreg2 to mmxreg1">
        <arg direction="input" type="mmxreg" />
        <arg direction="output" type="mmxreg" />
        <description>Bitwise Or</description>
    </op>
    <op bitmask="01100110000111111101011" mnemonic="POR" detail="xmemory to xmmreg">
        <arg direction="input" type="xmemory" />
        <arg direction="output" type="xmmreg" />
        <description>Bitwise Or</description>
```

```
    </op>
    <op bitmask="01100110000011111110101111" mnemonic="POR" detail="xmmreg2 to xmmreg1">
        <arg direction="input" type="xmmreg" />
        <arg direction="output" type="xmmreg" />
        <description>Bitwise Or</description>
    </op>
    <op bitmask="0000111100011000xx000" mnemonic="PREFETCHNTA">
        <arg direction="input" type="mem" />
        <arg direction="output" type="mem" />
        <description>Prefetch Non-Temporal to All Cache Levels</description>
    </op>
    <op bitmask="0000111100011000xx001" mnemonic="PREFETCHT0">
        <arg direction="input" type="mem" />
        <arg direction="output" type="mem" />
        <description>Prefetch Temporal to All Cache Levels</description>
    </op>
    <op bitmask="0000111100011000xx010" mnemonic="PREFETCHT1">
        <arg direction="input" type="mem" />
        <arg direction="output" type="mem" />
        <description>Prefetch Temporal to First Level Cache</description>
    </op>
    <op bitmask="0000111100011000xx011" mnemonic="PREFETCHT2">
        <arg direction="input" type="mem" />
        <arg direction="output" type="mem" />
        <description>Prefetch Temporal to Second Level Cache</description>
    </op>
    <op bitmask="0000111111110110" mnemonic="PSADBW" detail="mem to mmreg">
        <arg direction="input" type="mem" />
        <arg direction="output" type="mmreg" />
        <description>Compute Sum of Absolute Differences</description>
    </op>
    <op bitmask="0000111111111011" mnemonic="PSADBW" detail="mmreg to mmreg">
        <arg direction="input" type="mmreg" />
        <arg direction="output" type="mmreg" />
        <description>Compute Sum of Absolute Differences</description>
    </op>
    <op bitmask="01100110000111111110110" mnemonic="PSADBW" detail="mem to xmmreg">
        <arg direction="input" type="mem" />
        <arg direction="output" type="xmmreg" />
        <description>Compute Sum of Absolute Differences</description>
    </op>
    <op bitmask="01100110000011111111011011" mnemonic="PSADBW" detail="xmmreg to xmmreg">
        <arg direction="input" type="xmmreg" />
        <arg direction="output" type="xmmreg" />
        <description>Compute Sum of Absolute Differences</description>
    </op>
    <op bitmask="01100110000011111110111000011" mnemonic="PSHUFD" detail="xmmreg to xmmreg, imm
8, mem to xmmreg, imm8">
        <arg direction="input" type="mem" />
        <arg direction="input" type="imm" />
        <arg direction="output" type="xmmreg" />
        <description>Shuffle Packed Doublewords imm8</description>
    </op>
    <op bitmask="11110011000011111110111000011" mnemonic="PSHUFHW" detail="xmmreg to xmmreg, im
m8, mem to xmmreg, imm8">
        <arg direction="input" type="mem" />
        <arg direction="input" type="imm" />
        <arg direction="output" type="xmmreg" />
        <description>Shuffle Packed High Words</description>
    </op>
    <op bitmask="11110010000011111110111000011" mnemonic="PSHUFLW" detail="xmmreg to xmmreg, im
m8, mem to xmmreg, imm8">
        <arg direction="input" type="mem" />
        <arg direction="input" type="imm" />
        <arg direction="output" type="xmmreg" />
```

```
        <description>Shuffle Packed Low Words</description>
    </op>
    <op bitmask="000011110111000011" mnemonic="PSHUFW" detail="mmreg to mmreg, imm8, mem to
mmreg, imm8">
        <arg direction="input" type="mem" />
        <arg direction="input" type="imm" />
        <arg direction="output" type="mmreg" />
        <description>Shuffle Packed Words</description>
    </op>
    <op bitmask="01100110000011111011100xx11110" mnemonic="PSLL" detail="xmmreg by immediate"
>
        <arg direction="input" type="xmmreg" />
        <arg direction="input" type="imm" />
        <arg direction="output" type="xmmreg" />
        <description>Packed Shift Left Logical</description>
    </op>
    <op bitmask="0110011000001111111100" mnemonic="PSLL" detail="xmmreg by memory">
        <arg direction="input" type="xmmreg" />
        <arg direction="input" type="mem" />
        <arg direction="output" type="xmmreg" />
        <description>Packed Shift Left Logical</description>
    </op>
    <op bitmask="0110011000001111111100xx11" mnemonic="PSLL" detail="xmmreg1 by xmmreg2">
        <arg direction="input" type="xmmreg" />
        <arg direction="input" type="xmmreg" />
        <arg direction="output" type="xmmreg" />
        <description>Packed Shift Left Logical</description>
    </op>
    <op bitmask="000011110111100xx11110" mnemonic="PSLL2" detail="mmxreg by immediate">
        <arg direction="input" type="mmxreg" />
        <arg direction="input" type="imm" />
        <arg direction="output" type="mmxreg" />
        <description>Packed shift left logical</description>
    </op>
    <op bitmask="000011111111100" mnemonic="PSLL2" detail="mmxreg by memory">
        <arg direction="input" type="mmxreg" />
        <arg direction="input" type="mem" />
        <arg direction="output" type="mmxreg" />
        <description>Packed shift left logical</description>
    </op>
    <op bitmask="000011111111100xx11" mnemonic="PSLL2" detail="mmxreg1 by mmxreg2">
        <arg direction="input" type="mmxreg" />
        <arg direction="input" type="mmxreg" />
        <arg direction="output" type="mmxreg" />
        <description>Packed shift left logical</description>
    </op>
    <op bitmask="01100110000011110111001111111" mnemonic="PSLLDQ" detail="xmmreg, imm8">
        <arg direction="input" type="imm" />
        <arg direction="output" type="xmmreg" />
        <description>Shift Double Quadword Left Logical</description>
    </op>
    <op bitmask="01100110000011110111100xx11100" mnemonic="PSRA" detail="xmmreg by immediate"
>
        <arg direction="input" type="xmmreg" />
        <arg direction="input" type="imm" />
        <arg direction="output" type="xmmreg" />
        <description>Packed Shift Right Arithmetic</description>
    </op>
    <op bitmask="0110011000001111111000" mnemonic="PSRA" detail="xmmreg by memory">
        <arg direction="input" type="xmmreg" />
        <arg direction="input" type="mem" />
        <arg direction="output" type="xmmreg" />
        <description>Packed Shift Right Arithmetic</description>
    </op>
    <op bitmask="0110011000001111111000xx11" mnemonic="PSRA" detail="xmmreg1 by xmmreg2">
```

```
        <arg direction="input" type="xmmreg" />
        <arg direction="input" type="xmmreg" />
        <arg direction="output" type="xmmreg" />
        <description>Packed Shift Right Arithmetic</description>
    </op>
    <op bitmask="000011110111100xx11100" mnemonic="PSRA2" detail="mmxreg by immediate">
        <arg direction="input" type="mmxreg" />
        <arg direction="input" type="imm" />
        <arg direction="output" type="mmxreg" />
        <description>Packed shift right arithmetic</description>
    </op>
    <op bitmask="000011111111000" mnemonic="PSRA2" detail="mmxreg by memory">
        <arg direction="input" type="mmxreg" />
        <arg direction="input" type="mem" />
        <arg direction="output" type="mmxreg" />
        <description>Packed shift right arithmetic</description>
    </op>
    <op bitmask="000011111111000xx11" mnemonic="PSRA2" detail="mmxreg1 by mmxreg2">
        <arg direction="input" type="mmxreg" />
        <arg direction="input" type="mmxreg" />
        <arg direction="output" type="mmxreg" />
        <description>Packed shift right arithmetic</description>
    </op>
    <op bitmask="000011110111100xx11010" mnemonic="PSRL" detail="mmxreg by immediate">
        <arg direction="input" type="mmxreg" />
        <arg direction="input" type="imm" />
        <arg direction="output" type="mmxreg" />
        <description>Packed shift right logical</description>
    </op>
    <op bitmask="000011111110100" mnemonic="PSRL" detail="mmxreg by memory">
        <arg direction="input" type="mmxreg" />
        <arg direction="input" type="mem" />
        <arg direction="output" type="mmxreg" />
        <description>Packed shift right logical</description>
    </op>
    <op bitmask="000011111110100xx11" mnemonic="PSRL" detail="mmxreg1 by mmxreg2">
        <arg direction="input" type="mmxreg" />
        <arg direction="input" type="mmxreg" />
        <arg direction="output" type="mmxreg" />
        <description>Packed shift right logical</description>
    </op>
    <op bitmask="01100110000011110111100xx11010" mnemonic="PSRL" detail="xmmxreg by immediate
">
        <arg direction="input" type="xmmreg" />
        <arg direction="input" type="imm" />
        <arg direction="output" type="xmmreg" />
        <description>Packed Shift Right Logical</description>
    </op>
    <op bitmask="0110011000001111110100" mnemonic="PSRL" detail="xmmxreg by memory">
        <arg direction="input" type="xmmreg" />
        <arg direction="input" type="mem" />
        <arg direction="output" type="xmmreg" />
        <description>Packed Shift Right Logical</description>
    </op>
    <op bitmask="0110011000001111110100xx11" mnemonic="PSRL" detail="xmmxreg1 by xmmxreg2">
        <arg direction="input" type="xmmreg" />
        <arg direction="input" type="xmmreg" />
        <arg direction="output" type="xmmreg" />
        <description>Packed Shift Right Logical</description>
    </op>
    <op bitmask="01100110000011110111001111011" mnemonic="PSRLDQ" detail="xmmreg, imm8">
        <arg direction="input" type="imm" />
        <arg direction="output" type="xmmreg" />
        <description>Shift Double Quadword Right Logical</description>
    </op>
```

```
    <op bitmask="00001111111110" mnemonic="PSUB" detail="memory from mmxreg">
        <arg direction="input" type="mmxreg" />
        <arg direction="output" type="mem" />
        <description>Subtract with wrap-around</description>
    </op>
    <op bitmask="00001111111110xx11" mnemonic="PSUB" detail="mmxreg2 from mmxreg1">
        <arg direction="input" type="mmxreg" />
        <arg direction="output" type="mmxreg" />
        <description>Subtract with wrap-around</description>
    </op>
    <op bitmask="01100110000011111111110" mnemonic="PSUB" detail="memory from xmmreg">
        <arg direction="input" type="xmmreg" />
        <arg direction="output" type="mem" />
        <description>Subtract With Wrap-around</description>
    </op>
    <op bitmask="01100110000011111111110xx11" mnemonic="PSUB" detail="xmmreg2 from xmmreg1">
        <arg direction="input" type="xmmreg" />
        <arg direction="output" type="xmmreg" />
        <description>Subtract With Wrap-around</description>
    </op>
    <op bitmask="0000111111111011" mnemonic="PSUBQ" detail="mem to mmreg">
        <arg direction="input" type="mem" />
        <arg direction="output" type="mmreg" />
        <description>Subtract Packed Quadword Integers</description>
    </op>
    <op bitmask="0000111111111101111" mnemonic="PSUBQ" detail="mmreg to mmreg">
        <arg direction="input" type="mmreg" />
        <arg direction="output" type="mmreg" />
        <description>Subtract Packed Quadword Integers</description>
    </op>
    <op bitmask="01100110000011111111011" mnemonic="PSUBQ" detail="mem to xmmreg">
        <arg direction="input" type="mem" />
        <arg direction="output" type="xmmreg" />
        <description>Subtract Packed Quadword Integers</description>
    </op>
    <op bitmask="011001100000111111111101111" mnemonic="PSUBQ" detail="xmmreg to xmmreg">
        <arg direction="input" type="xmmreg" />
        <arg direction="output" type="xmmreg" />
        <description>Subtract Packed Quadword Integers</description>
    </op>
    <op bitmask="00001111111010" mnemonic="PSUBS" detail="memory from mmxreg">
        <arg direction="input" type="mmxreg" />
        <arg direction="output" type="mem" />
        <description>Subtract signed with saturation</description>
    </op>
    <op bitmask="00001111111010xx11" mnemonic="PSUBS" detail="mmxreg2 from mmxreg1">
        <arg direction="input" type="mmxreg" />
        <arg direction="output" type="mmxreg" />
        <description>Subtract signed with saturation</description>
    </op>
    <op bitmask="01100110000011111111010" mnemonic="PSUBS" detail="memory from xmmreg">
        <arg direction="input" type="xmmreg" />
        <arg direction="output" type="mem" />
        <description>Subtract Signed With Saturation</description>
    </op>
    <op bitmask="01100110000011111111010xx11" mnemonic="PSUBS" detail="xmmreg2 from xmmreg1">
        <arg direction="input" type="xmmreg" />
        <arg direction="output" type="xmmreg" />
        <description>Subtract Signed With Saturation</description>
    </op>
    <op bitmask="00001111110110" mnemonic="PSUBUS" detail="memory from mmxreg, memory from x
mmreg">
        <arg direction="input" type="xmmreg" />
        <arg direction="output" type="mem" />
        <description>Subtract Unsigned With Saturation</description>
```

```
    </op>
    <op bitmask="00001111110110xx11" mnemonic="PSUBUS" detail="mmxreg2 from mmxreg1, xmmreg2
 from xmmreg1">
        <arg direction="input" type="xmmreg" />
        <arg direction="output" type="xmmreg" />
        <description>Subtract Unsigned With Saturation</description>
    </op>
    <op bitmask="00001111011010" mnemonic="PUNPCKH" detail="memory to mmxreg">
        <arg direction="input" type="mem" />
        <arg direction="output" type="mmxreg" />
        <description>Unpack high data to next larger type</description>
    </op>
    <op bitmask="00001111011010xx11" mnemonic="PUNPCKH" detail="mmxreg2 to mmxreg1">
        <arg direction="input" type="mmxreg" />
        <arg direction="output" type="mmxreg" />
        <description>Unpack high data to next larger type</description>
    </op>
    <op bitmask="01100110000011111011010" mnemonic="PUNPCKH" detail="mem to xmmreg">
        <arg direction="input" type="mem" />
        <arg direction="output" type="xmmreg" />
        <description>Unpack High Data To Next Larger Type</description>
    </op>
    <op bitmask="01100110000011111011010xx11" mnemonic="PUNPCKH" detail="xmmreg to xmmreg">
        <arg direction="input" type="xmmreg" />
        <arg direction="output" type="xmmreg" />
        <description>Unpack High Data To Next Larger Type</description>
    </op>
    <op bitmask="01100110000011111011101101" mnemonic="PUNPCKHQDQ" detail="mem to xmmreg">
        <arg direction="input" type="mem" />
        <arg direction="output" type="xmmreg" />
        <description>Unpack High Data</description>
    </op>
    <op bitmask="01100110000011111011011011 1" mnemonic="PUNPCKHQDQ" detail="xmmreg to xmmreg"
>
        <arg direction="input" type="xmmreg" />
        <arg direction="output" type="xmmreg" />
        <description>Unpack High Data</description>
    </op>
    <op bitmask="00001111011000" mnemonic="PUNPCKL" detail="memory to mmxreg">
        <arg direction="input" type="mem" />
        <arg direction="output" type="mmxreg" />
        <description>Unpack low data to next larger type</description>
    </op>
    <op bitmask="00001111011000xx11" mnemonic="PUNPCKL" detail="mmxreg2 to mmxreg1">
        <arg direction="input" type="mmxreg" />
        <arg direction="output" type="mmxreg" />
        <description>Unpack low data to next larger type</description>
    </op>
    <op bitmask="01100110000011111011000" mnemonic="PUNPCKL" detail="mem to xmmreg">
        <arg direction="input" type="mem" />
        <arg direction="output" type="xmmreg" />
        <description>Unpack Low Data To Next Larger Type</description>
    </op>
    <op bitmask="01100110000011111011000xx11" mnemonic="PUNPCKL" detail="xmmreg to xmmreg">
        <arg direction="input" type="xmmreg" />
        <arg direction="output" type="xmmreg" />
        <description>Unpack Low Data To Next Larger Type</description>
    </op>
    <op bitmask="01100110000011111011101100" mnemonic="PUNPCKLQDQ" detail="mem to xmmreg">
        <arg direction="input" type="mem" />
        <arg direction="output" type="xmmreg" />
        <description>Unpack Low Data</description>
    </op>
    <op bitmask="01100110000011111011011011 " mnemonic="PUNPCKLQDQ" detail="xmmreg to xmmreg"
>
```

```
        <arg direction="input" type="xmmreg" />
        <arg direction="output" type="xmmreg" />
        <description>Unpack Low Data</description>
    </op>
    <op bitmask="0000111110xx000" mnemonic="PUSH" detail="segment register FS,GS">
        <arg direction="input" type="SRx" />
        <description>Push Segment Register onto the Stack</description>
    </op>
    <op bitmask="000xx110" mnemonic="PUSH" detail="segment register CS,DS,ES,SS">
        <arg direction="input" type="SRx" />
        <description>Push Segment Register onto the Stack</description>
    </op>
    <op bitmask="01010" mnemonic="PUSH" detail="register (alternate encoding)">
        <arg direction="input" type="reg" />
        <description>Push Operand onto the Stack</description>
    </op>
    <op bitmask="011010x0" mnemonic="PUSH" detail="immediate">
        <arg direction="input" type="imm" />
        <description>Push Operand onto the Stack</description>
    </op>
    <op bitmask="1111111111110" mnemonic="PUSH" detail="register">
        <arg direction="input" type="reg" />
        <description>Push Operand onto the Stack</description>
    </op>
    <op bitmask="11111111xx110" mnemonic="PUSH" detail="memory">
        <arg direction="input" type="mem" />
        <description>Push Operand onto the Stack</description>
    </op>
    <op bitmask="01100000" mnemonic="PUSHA/PUSHAD">
        <description>Push All General Registers</description>
    </op>
    <op bitmask="10011100" mnemonic="PUSHF/PUSHFD">
        <description>Push Flags Register onto the Stack</description>
    </op>
    <op bitmask="0000111111101111" mnemonic="PXOR" detail="memory to mmxreg">
        <arg direction="input" type="mem" />
        <arg direction="output" type="mmxreg" />
        <description>Bitwise Xor</description>
    </op>
    <op bitmask="0000111111110111111" mnemonic="PXOR" detail="mmxreg2 to mmxreg1">
        <arg direction="input" type="mmxreg" />
        <arg direction="output" type="mmxreg" />
        <description>Bitwise Xor</description>
    </op>
    <op bitmask="011001100000111111101111" mnemonic="PXOR" detail="memory to xmmreg">
        <arg direction="input" type="mem" />
        <arg direction="output" type="xmmreg" />
        <description>Bitwise Xor</description>
    </op>
    <op bitmask="01100110000011111110111111" mnemonic="PXOR" detail="xmmreg2 to xmmreg1">
        <arg direction="input" type="xmmreg" />
        <arg direction="output" type="xmmreg" />
        <description>Bitwise Xor</description>
    </op>
    <op bitmask="1100000x11010" mnemonic="RCL" detail="register by immediate count">
        <arg direction="input" type="reg" />
        <arg direction="input" type="imm" />
        <arg direction="output" type="reg" />
        <description>Rotate thru Carry Left</description>
    </op>
    <op bitmask="1100000xxx010" mnemonic="RCL" detail="memory by immediate count">
        <arg direction="input" type="mem" />
        <arg direction="input" type="imm" />
        <arg direction="output" type="mem" />
        <description>Rotate thru Carry Left</description>
```

```
    </op>
    <op bitmask="1101000x11010" mnemonic="RCL" detail="register by 1">
        <arg direction="input" type="reg" />
        <arg direction="output" type="reg" />
        <description>Rotate thru Carry Left</description>
    </op>
    <op bitmask="1101000xxx010" mnemonic="RCL" detail="memory by 1">
        <arg direction="input" type="mem" />
        <arg direction="output" type="mem" />
        <description>Rotate thru Carry Left</description>
    </op>
    <op bitmask="1101001x11010" mnemonic="RCL" detail="register by CL">
        <arg direction="input" type="reg" />
        <arg direction="input" type="reg" />
        <arg direction="output" type="reg" />
        <description>Rotate thru Carry Left</description>
    </op>
    <op bitmask="1101001xxx010" mnemonic="RCL" detail="memory by CL">
        <arg direction="input" type="mem" />
        <arg direction="input" type="reg" />
        <arg direction="output" type="mem" />
        <description>Rotate thru Carry Left</description>
    </op>
    <op bitmask="0000111101010011" mnemonic="RCPPS" detail="mem to xmmreg">
        <arg direction="input" type="mem" />
        <arg direction="output" type="xmmreg" />
        <description>Compute Reciprocals of Packed Single-Precision Floating-Point Values</d
escription>
    </op>
    <op bitmask="0000111110101001111" mnemonic="RCPPS" detail="xmmreg to xmmreg">
        <arg direction="input" type="xmmreg" />
        <arg direction="output" type="xmmreg" />
        <description>Compute Reciprocals of Packed Single-Precision Floating-Point Values</d
escription>
    </op>
    <op bitmask="11110011000011110101010011" mnemonic="RCPSS" detail="mem to xmmreg">
        <arg direction="input" type="mem" />
        <arg direction="output" type="xmmreg" />
        <description>Compute Reciprocals of Scalar Single-Precision Floating-Point Value</de
scription>
    </op>
    <op bitmask="11110011000011110101001111" mnemonic="RCPSS" detail="xmmreg to xmmreg">
        <arg direction="input" type="xmmreg" />
        <arg direction="output" type="xmmreg" />
        <description>Compute Reciprocals of Scalar Single-Precision Floating-Point Value</de
scription>
    </op>
    <op bitmask="1100000x11011" mnemonic="RCR" detail="register by immediate count">
        <arg direction="input" type="reg" />
        <arg direction="input" type="imm" />
        <arg direction="output" type="reg" />
        <description>Rotate thru Carry Right</description>
    </op>
    <op bitmask="1100000xxx011" mnemonic="RCR" detail="memory by immediate count">
        <arg direction="input" type="mem" />
        <arg direction="input" type="imm" />
        <arg direction="output" type="mem" />
        <description>Rotate thru Carry Right</description>
    </op>
    <op bitmask="1101000x11011" mnemonic="RCR" detail="register by 1">
        <arg direction="input" type="reg" />
        <arg direction="output" type="reg" />
        <description>Rotate thru Carry Right</description>
    </op>
    <op bitmask="1101000xxx011" mnemonic="RCR" detail="memory by 1">
```

```
        <arg direction="input" type="mem" />
        <arg direction="output" type="mem" />
        <description>Rotate thru Carry Right</description>
    </op>
    <op bitmask="1101001x11011" mnemonic="RCR" detail="register by CL">
        <arg direction="input" type="reg" />
        <arg direction="input" type="reg" />
        <arg direction="output" type="reg" />
        <description>Rotate thru Carry Right</description>
    </op>
    <op bitmask="1101001xxx011" mnemonic="RCR" detail="memory by CL">
        <arg direction="input" type="mem" />
        <arg direction="input" type="reg" />
        <arg direction="output" type="mem" />
        <description>Rotate thru Carry Right</description>
    </op>
    <op bitmask="0000111100110010" mnemonic="RDMSR">
        <description>Read from Model-Specific Register</description>
    </op>
    <op bitmask="0000111100110011" mnemonic="RDPMC">
        <description>Read Performance Monitoring Counters</description>
    </op>
    <op bitmask="0000111100110001" mnemonic="RDTSC">
        <description>Read Time-Stamp Counter</description>
    </op>
    <op bitmask="111100110110110" mnemonic="REP INS">
        <description>Input String</description>
    </op>
    <op bitmask="111100111010110" mnemonic="REP LODS">
        <description>Load String</description>
    </op>
    <op bitmask="111100111010010" mnemonic="REP MOVS">
        <description>Move String</description>
    </op>
    <op bitmask="111100110110111" mnemonic="REP OUTS">
        <description>Output String</description>
    </op>
    <op bitmask="111100111010101" mnemonic="REP STOS">
        <description>Store String</description>
    </op>
    <op bitmask="111100111010011" mnemonic="REPE CMPS">
        <description>Compare String</description>
    </op>
    <op bitmask="111100111010111" mnemonic="REPE SCAS">
        <description>Scan String</description>
    </op>
    <op bitmask="111100101010011" mnemonic="REPNE CMPS">
        <description>Compare String</description>
    </op>
    <op bitmask="111100101010111" mnemonic="REPNE SCAS">
        <description>Scan String</description>
    </op>
    <op bitmask="11000010" mnemonic="RET" detail="adding immediate to SP">
        <arg direction="input" type="imm" />
        <arg direction="input" type="reg" />
        <description>Return from Procedure (to same segment)</description>
    </op>
    <op bitmask="11000011" mnemonic="RET" detail="no argument">
        <description>Return from Procedure (to same segment)</description>
    </op>
    <op bitmask="11001010" mnemonic="RET" detail="adding immediate to SP">
        <arg direction="input" type="imm" />
        <arg direction="input" type="reg" />
        <description>Return from Procedure (to other segment)</description>
    </op>
```

```
    <op bitmask="11001011" mnemonic="RET" detail="intersegment">
        <description>Return from Procedure (to other segment)</description>
    </op>
    <op bitmask="1100000x11000" mnemonic="ROL" detail="register by immediate count">
        <arg direction="input" type="reg" />
        <arg direction="input" type="imm" />
        <arg direction="output" type="reg" />
        <description>Rotate Left</description>
    </op>
    <op bitmask="1100000xxx000" mnemonic="ROL" detail="memory by immediate count">
        <arg direction="input" type="mem" />
        <arg direction="input" type="imm" />
        <arg direction="output" type="mem" />
        <description>Rotate Left</description>
    </op>
    <op bitmask="1101000x11000" mnemonic="ROL" detail="register by 1">
        <arg direction="input" type="reg" />
        <arg direction="output" type="reg" />
        <description>Rotate Left</description>
    </op>
    <op bitmask="1101000xxx000" mnemonic="ROL" detail="memory by 1">
        <arg direction="input" type="mem" />
        <arg direction="output" type="mem" />
        <description>Rotate Left</description>
    </op>
    <op bitmask="1101001x11000" mnemonic="ROL" detail="register by CL">
        <arg direction="input" type="reg" />
        <arg direction="input" type="reg" />
        <arg direction="output" type="reg" />
        <description>Rotate Left</description>
    </op>
    <op bitmask="1101001xxx000" mnemonic="ROL" detail="memory by CL">
        <arg direction="input" type="mem" />
        <arg direction="input" type="reg" />
        <arg direction="output" type="mem" />
        <description>Rotate Left</description>
    </op>
    <op bitmask="1100000x11001" mnemonic="ROR" detail="register by immediate count">
        <arg direction="input" type="reg" />
        <arg direction="input" type="imm" />
        <arg direction="output" type="reg" />
        <description>Rotate Right</description>
    </op>
    <op bitmask="1100000xxx001" mnemonic="ROR" detail="memory by immediate count">
        <arg direction="input" type="mem" />
        <arg direction="input" type="imm" />
        <arg direction="output" type="mem" />
        <description>Rotate Right</description>
    </op>
    <op bitmask="1101000x11001" mnemonic="ROR" detail="register by 1">
        <arg direction="input" type="reg" />
        <arg direction="output" type="reg" />
        <description>Rotate Right</description>
    </op>
    <op bitmask="1101000xxx001" mnemonic="ROR" detail="memory by 1">
        <arg direction="input" type="mem" />
        <arg direction="output" type="mem" />
        <description>Rotate Right</description>
    </op>
    <op bitmask="1101001x11001" mnemonic="ROR" detail="register by CL">
        <arg direction="input" type="reg" />
        <arg direction="input" type="reg" />
        <arg direction="output" type="reg" />
        <description>Rotate Right</description>
    </op>
```

```
    <op bitmask="1101001xxx001" mnemonic="ROR" detail="memory by CL">
        <arg direction="input" type="mem" />
        <arg direction="input" type="reg" />
        <arg direction="output" type="mem" />
        <description>Rotate Right</description>
    </op>
    <op bitmask="0000111110101010" mnemonic="RSM">
        <description>Resume from System Management Mode</description>
    </op>
    <op bitmask="0000111101010010" mnemonic="RSQRTPS" detail="mem to xmmreg">
        <arg direction="input" type="mem" />
        <arg direction="output" type="xmmreg" />
        <description>Compute Reciprocals of Square Roots of Packed Single-Precision Floating
-Point Values</description>
    </op>
    <op bitmask="0000111101001011" mnemonic="RSQRTPS" detail="xmmreg to xmmreg">
        <arg direction="input" type="xmmreg" />
        <arg direction="output" type="xmmreg" />
        <description>Compute Reciprocals of Square Roots of Packed Single-Precision Floating
-Point Values</description>
    </op>
    <op bitmask="11110011000011110101010010" mnemonic="RSQRTSS" detail="mem to xmmreg">
        <arg direction="input" type="mem" />
        <arg direction="output" type="xmmreg" />
        <description>Compute Reciprocals of Square Roots of Scalar Single-Precision Floating
-Point Value</description>
    </op>
    <op bitmask="11110011000011110101001011" mnemonic="RSQRTSS" detail="xmmreg to xmmreg">
        <arg direction="input" type="xmmreg" />
        <arg direction="output" type="xmmreg" />
        <description>Compute Reciprocals of Square Roots of Scalar Single-Precision Floating
-Point Value</description>
    </op>
    <op bitmask="10011110" mnemonic="SAHF">
        <arg direction="output" type="Flags" />
        <description>Store AH into Flags</description>
    </op>
    <op bitmask="1100000x11111" mnemonic="SAR" detail="register by immediate count">
        <arg direction="input" type="reg" />
        <arg direction="input" type="imm" />
        <arg direction="output" type="reg" />
        <description>Shift Arithmetic Right</description>
    </op>
    <op bitmask="1100000xxx111" mnemonic="SAR" detail="memory by immediate count">
        <arg direction="input" type="mem" />
        <arg direction="input" type="imm" />
        <arg direction="output" type="mem" />
        <description>Shift Arithmetic Right</description>
    </op>
    <op bitmask="1101000x11111" mnemonic="SAR" detail="register by 1">
        <arg direction="input" type="reg" />
        <arg direction="output" type="reg" />
        <description>Shift Arithmetic Right</description>
    </op>
    <op bitmask="1101000xxx111" mnemonic="SAR" detail="memory by 1">
        <arg direction="input" type="mem" />
        <arg direction="output" type="mem" />
        <description>Shift Arithmetic Right</description>
    </op>
    <op bitmask="1101001x11111" mnemonic="SAR" detail="register by CL">
        <arg direction="input" type="reg" />
        <arg direction="input" type="reg" />
        <arg direction="output" type="reg" />
        <description>Shift Arithmetic Right</description>
    </op>
```

```
    <op bitmask="1101001xxx111" mnemonic="SAR" detail="memory by CL">
        <arg direction="input" type="mem" />
        <arg direction="input" type="reg" />
        <arg direction="output" type="mem" />
        <description>Shift Arithmetic Right</description>
    </op>
    <op bitmask="0001100" mnemonic="SBB" detail="register to memory">
        <arg direction="input" type="reg" />
        <arg direction="output" type="mem" />
        <description>Integer Subtraction with Borrow</description>
    </op>
    <op bitmask="0001100x11" mnemonic="SBB" detail="register1 to register2">
        <arg direction="input" type="reg" />
        <arg direction="output" type="reg" />
        <description>Integer Subtraction with Borrow</description>
    </op>
    <op bitmask="0001101" mnemonic="SBB" detail="memory to register">
        <arg direction="input" type="mem" />
        <arg direction="output" type="reg" />
        <description>Integer Subtraction with Borrow</description>
    </op>
    <op bitmask="0001101x11" mnemonic="SBB" detail="register2 to register1">
        <arg direction="input" type="reg" />
        <arg direction="output" type="reg" />
        <description>Integer Subtraction with Borrow</description>
    </op>
    <op bitmask="0001110" mnemonic="SBB" detail="immediate to AL, AX, or EAX">
        <arg direction="input" type="imm" />
        <arg direction="output" type="reg" />
        <description>Integer Subtraction with Borrow</description>
    </op>
    <op bitmask="100000xx11011" mnemonic="SBB" detail="immediate to register">
        <arg direction="input" type="imm" />
        <arg direction="output" type="reg" />
        <description>Integer Subtraction with Borrow</description>
    </op>
    <op bitmask="100000xxxx011" mnemonic="SBB" detail="immediate to memory">
        <arg direction="input" type="imm" />
        <arg direction="output" type="mem" />
        <description>Integer Subtraction with Borrow</description>
    </op>
    <op bitmask="1010111" mnemonic="SCAS/SCASB/SCASW/SCASD">
        <description>Scan String</description>
    </op>
    <op bitmask="00001111001000011000" mnemonic="SETcc" detail="register" conditional="O">
        <arg direction="output" type="reg" />
        <description>Byte Set on Condition: Overflow</description>
    </op>
    <op bitmask="0000111110010000xx000" mnemonic="SETcc" detail="memory" conditional="O">
        <arg direction="output" type="mem" />
        <description>Byte Set on Condition: Overflow</description>
    </op>
    <op bitmask="00001111001000111000" mnemonic="SETcc" detail="register" conditional="NO">
        <arg direction="output" type="reg" />
        <description>Byte Set on Condition: No overflow</description>
    </op>
    <op bitmask="0000111110010001xx000" mnemonic="SETcc" detail="memory" conditional="NO">
        <arg direction="output" type="mem" />
        <description>Byte Set on Condition: No overflow</description>
    </op>
    <op bitmask="00001111001001011000" mnemonic="SETcc" detail="register" conditional="B">
        <arg direction="output" type="reg" />
        <description>Byte Set on Condition: Below</description>
    </op>
    <op bitmask="0000111110010010xx000" mnemonic="SETcc" detail="memory" conditional="B">
```

```
        <arg direction="output" type="mem" />
        <description>Byte Set on Condition: Below</description>
    </op>
    <op bitmask="0000111110001001111000" mnemonic="SETcc" detail="register" conditional="NB">
        <arg direction="output" type="reg" />
        <description>Byte Set on Condition: Not below</description>
    </op>
    <op bitmask="0000111110010011xx000" mnemonic="SETcc" detail="memory" conditional="NB">
        <arg direction="output" type="mem" />
        <description>Byte Set on Condition: Not below</description>
    </op>
    <op bitmask="0000111110001010011000" mnemonic="SETcc" detail="register" conditional="E">
        <arg direction="output" type="reg" />
        <description>Byte Set on Condition: Equals</description>
    </op>
    <op bitmask="0000111110010100xx000" mnemonic="SETcc" detail="memory" conditional="E">
        <arg direction="output" type="mem" />
        <description>Byte Set on Condition: Equals</description>
    </op>
    <op bitmask="0000111110001010111000" mnemonic="SETcc" detail="register" conditional="NE">
        <arg direction="output" type="reg" />
        <description>Byte Set on Condition: Not equals</description>
    </op>
    <op bitmask="0000111110010101xx000" mnemonic="SETcc" detail="memory" conditional="NE">
        <arg direction="output" type="mem" />
        <description>Byte Set on Condition: Not equals</description>
    </op>
    <op bitmask="0000111110001011011000" mnemonic="SETcc" detail="register" conditional="NA">
        <arg direction="output" type="reg" />
        <description>Byte Set on Condition: Not above</description>
    </op>
    <op bitmask="0000111110010110xx000" mnemonic="SETcc" detail="memory" conditional="NA">
        <arg direction="output" type="mem" />
        <description>Byte Set on Condition: Not above</description>
    </op>
    <op bitmask="0000111110001011111000" mnemonic="SETcc" detail="register" conditional="A">
        <arg direction="output" type="reg" />
        <description>Byte Set on Condition: Above</description>
    </op>
    <op bitmask="0000111110010111xx000" mnemonic="SETcc" detail="memory" conditional="A">
        <arg direction="output" type="mem" />
        <description>Byte Set on Condition: Above</description>
    </op>
    <op bitmask="0000111110001100011000" mnemonic="SETcc" detail="register" conditional="S">
        <arg direction="output" type="reg" />
        <description>Byte Set on Condition: Sign</description>
    </op>
    <op bitmask="0000111110011000xx000" mnemonic="SETcc" detail="memory" conditional="S">
        <arg direction="output" type="mem" />
        <description>Byte Set on Condition: Sign</description>
    </op>
    <op bitmask="0000111110001100111000" mnemonic="SETcc" detail="register" conditional="NS">
        <arg direction="output" type="reg" />
        <description>Byte Set on Condition: Not sign</description>
    </op>
    <op bitmask="0000111110011001xx000" mnemonic="SETcc" detail="memory" conditional="NS">
        <arg direction="output" type="mem" />
        <description>Byte Set on Condition: Not sign</description>
    </op>
    <op bitmask="0000111110001101011000" mnemonic="SETcc" detail="register" conditional="P">
        <arg direction="output" type="reg" />
        <description>Byte Set on Condition: Parity</description>
    </op>
    <op bitmask="0000111110011010xx000" mnemonic="SETcc" detail="memory" conditional="P">
        <arg direction="output" type="mem" />
```

```
        <description>Byte Set on Condition: Parity</description>
    </op>
    <op bitmask="0000111110011011111000" mnemonic="SETcc" detail="register" conditional="NP">
        <arg direction="output" type="reg" />
        <description>Byte Set on Condition: Not parity</description>
    </op>
    <op bitmask="0000111110011011xx000" mnemonic="SETcc" detail="memory" conditional="NP">
        <arg direction="output" type="mem" />
        <description>Byte Set on Condition: Not parity</description>
    </op>
    <op bitmask="0000111110011110011000" mnemonic="SETcc" detail="register" conditional="L">
        <arg direction="output" type="reg" />
        <description>Byte Set on Condition: Less than</description>
    </op>
    <op bitmask="0000111110011100xx000" mnemonic="SETcc" detail="memory" conditional="L">
        <arg direction="output" type="mem" />
        <description>Byte Set on Condition: Less than</description>
    </op>
    <op bitmask="0000111110011101111000" mnemonic="SETcc" detail="register" conditional="NL">
        <arg direction="output" type="reg" />
        <description>Byte Set on Condition: Not less than</description>
    </op>
    <op bitmask="0000111110011101xx000" mnemonic="SETcc" detail="memory" conditional="NL">
        <arg direction="output" type="mem" />
        <description>Byte Set on Condition: Not less than</description>
    </op>
    <op bitmask="0000111110011111011000" mnemonic="SETcc" detail="register" conditional="NG">
        <arg direction="output" type="reg" />
        <description>Byte Set on Condition: Not greater than</description>
    </op>
    <op bitmask="0000111110011111xx000" mnemonic="SETcc" detail="memory" conditional="NG">
        <arg direction="output" type="mem" />
        <description>Byte Set on Condition: Not greater than</description>
    </op>
    <op bitmask="0000111110011111111000" mnemonic="SETcc" detail="register" conditional="G">
        <arg direction="output" type="reg" />
        <description>Byte Set on Condition: Greater than</description>
    </op>
    <op bitmask="0000111110011111xx000" mnemonic="SETcc" detail="memory" conditional="G">
        <arg direction="output" type="mem" />
        <description>Byte Set on Condition: Greater than</description>
    </op>
    <op bitmask="000011111010111011111000" mnemonic="SFENCE">
        <description>Store Fence</description>
    </op>
    <op bitmask="0000111100000001xx000" mnemonic="SGDT">
        <description>Store Global Descriptor Table Register</description>
    </op>
    <op bitmask="1100000x11100" mnemonic="SHL" detail="register by immediate count">
        <arg direction="input" type="reg" />
        <arg direction="input" type="imm" />
        <arg direction="output" type="reg" />
        <description>Shift Left</description>
    </op>
    <op bitmask="1100000xxx100" mnemonic="SHL" detail="memory by immediate count">
        <arg direction="input" type="mem" />
        <arg direction="input" type="imm" />
        <arg direction="output" type="mem" />
        <description>Shift Left</description>
    </op>
    <op bitmask="1101000x11100" mnemonic="SHL" detail="register by 1">
        <arg direction="input" type="reg" />
        <arg direction="output" type="reg" />
        <description>Shift Left</description>
    </op>
```

```
    <op bitmask="1101000xxx100" mnemonic="SHL" detail="memory by 1">
        <arg direction="input" type="mem" />
        <arg direction="output" type="mem" />
        <description>Shift Left</description>
    </op>
    <op bitmask="1101001x11100" mnemonic="SHL" detail="register by CL">
        <arg direction="input" type="reg" />
        <arg direction="input" type="reg" />
        <arg direction="output" type="reg" />
        <description>Shift Left</description>
    </op>
    <op bitmask="1101001xxx100" mnemonic="SHL" detail="memory by CL">
        <arg direction="input" type="mem" />
        <arg direction="input" type="reg" />
        <arg direction="output" type="mem" />
        <description>Shift Left</description>
    </op>
    <op bitmask="0000111110100100" mnemonic="SHLD" detail="memory by immediate count">
        <arg direction="input" type="mem" />
        <arg direction="input" type="imm" />
        <arg direction="output" type="mem" />
        <description>Double Precision Shift Left</description>
    </op>
    <op bitmask="0000111110010011" mnemonic="SHLD" detail="register by immediate count">
        <arg direction="input" type="reg" />
        <arg direction="input" type="imm" />
        <arg direction="output" type="reg" />
        <description>Double Precision Shift Left</description>
    </op>
    <op bitmask="0000111110100101" mnemonic="SHLD" detail="memory by CL">
        <arg direction="input" type="mem" />
        <arg direction="input" type="reg" />
        <arg direction="output" type="mem" />
        <description>Double Precision Shift Left</description>
    </op>
    <op bitmask="0000111110010111" mnemonic="SHLD" detail="register by CL">
        <arg direction="input" type="reg" />
        <arg direction="input" type="reg" />
        <arg direction="output" type="reg" />
        <description>Double Precision Shift Left</description>
    </op>
    <op bitmask="1100000x11101" mnemonic="SHR" detail="register by immediate count">
        <arg direction="input" type="reg" />
        <arg direction="input" type="imm" />
        <arg direction="output" type="reg" />
        <description>Shift Right</description>
    </op>
    <op bitmask="1100000xxx101" mnemonic="SHR" detail="memory by immediate count">
        <arg direction="input" type="mem" />
        <arg direction="input" type="imm" />
        <arg direction="output" type="mem" />
        <description>Shift Right</description>
    </op>
    <op bitmask="1101000x11101" mnemonic="SHR" detail="register by 1">
        <arg direction="input" type="reg" />
        <arg direction="output" type="reg" />
        <description>Shift Right</description>
    </op>
    <op bitmask="1101000xxx101" mnemonic="SHR" detail="memory by 1">
        <arg direction="input" type="mem" />
        <arg direction="output" type="mem" />
        <description>Shift Right</description>
    </op>
    <op bitmask="1101001x11101" mnemonic="SHR" detail="register by CL">
        <arg direction="input" type="reg" />
```

```
        <arg direction="input" type="reg" />
        <arg direction="output" type="reg" />
        <description>Shift Right</description>
    </op>
    <op bitmask="1101001xxx101" mnemonic="SHR" detail="memory by CL">
        <arg direction="input" type="mem" />
        <arg direction="input" type="reg" />
        <arg direction="output" type="mem" />
        <description>Shift Right</description>
    </op>
    <op bitmask="0000111110101100" mnemonic="SHRD" detail="memory by immediate count">
        <arg direction="input" type="mem" />
        <arg direction="input" type="imm" />
        <arg direction="output" type="mem" />
        <description>Double Precision Shift Right</description>
    </op>
    <op bitmask="0000111110110011" mnemonic="SHRD" detail="register by immediate count">
        <arg direction="input" type="reg" />
        <arg direction="input" type="imm" />
        <arg direction="output" type="reg" />
        <description>Double Precision Shift Right</description>
    </op>
    <op bitmask="0000111110101101" mnemonic="SHRD" detail="memory by CL">
        <arg direction="input" type="mem" />
        <arg direction="input" type="reg" />
        <arg direction="output" type="mem" />
        <description>Double Precision Shift Right</description>
    </op>
    <op bitmask="0000111110110111" mnemonic="SHRD" detail="register by CL">
        <arg direction="input" type="reg" />
        <arg direction="input" type="reg" />
        <arg direction="output" type="reg" />
        <description>Double Precision Shift Right</description>
    </op>
    <op bitmask="011001100000111111000110" mnemonic="SHUFPD" detail="mem to xmmreg, imm8">
        <arg direction="input" type="mem" />
        <arg direction="input" type="imm" />
        <arg direction="output" type="xmmreg" />
        <description>Shuffle Packed Double-Precision Floating-Point Values imm8</description
>
    </op>
    <op bitmask="011001100000111111100011011" mnemonic="SHUFPD" detail="xmmreg to xmmreg, imm
8">
        <arg direction="input" type="xmmreg" />
        <arg direction="input" type="imm" />
        <arg direction="output" type="xmmreg" />
        <description>Shuffle Packed Double-Precision Floating-Point Values imm8</description
>
    </op>
    <op bitmask="0000111111000110" mnemonic="SHUFPS" detail="mem to xmmreg, imm8">
        <arg direction="input" type="mem" />
        <arg direction="input" type="imm" />
        <arg direction="output" type="xmmreg" />
        <description>Shuffle Packed Single-Precision Floating-Point Values</description>
    </op>
    <op bitmask="0000111111100011011" mnemonic="SHUFPS" detail="xmmreg to xmmreg, imm8">
        <arg direction="input" type="xmmreg" />
        <arg direction="input" type="imm" />
        <arg direction="output" type="xmmreg" />
        <description>Shuffle Packed Single-Precision Floating-Point Values</description>
    </op>
    <op bitmask="0000111100000001xx001" mnemonic="SIDT">
        <description>Store Interrupt Descriptor Table Register</description>
    </op>
    <op bitmask="0000111100000000011000" mnemonic="SLDT" detail="to register">
```

```
        <arg direction="output" type="reg" />
        <description>Store Local Descriptor Table Register</description>
    </op>
    <op bitmask="0000111100000000xx000" mnemonic="SLDT" detail="to memory">
        <arg direction="output" type="mem" />
        <description>Store Local Descriptor Table Register</description>
    </op>
    <op bitmask="0000111100000000111100" mnemonic="SMSW" detail="to register">
        <arg direction="output" type="reg" />
        <description>Store Machine Status Word</description>
    </op>
    <op bitmask="0000111100000001xx100" mnemonic="SMSW" detail="to memory">
        <arg direction="output" type="mem" />
        <description>Store Machine Status Word</description>
    </op>
    <op bitmask="011001100000111101010001" mnemonic="SQRTPD" detail="mem to xmmreg">
        <arg direction="input" type="mem" />
        <arg direction="output" type="xmmreg" />
        <description>Compute Square Roots of Packed Double-Precision Floating-Point Values</
description>
    </op>
    <op bitmask="0110011000001111101010001111" mnemonic="SQRTPD" detail="xmmreg to xmmreg">
        <arg direction="input" type="xmmreg" />
        <arg direction="output" type="xmmreg" />
        <description>Compute Square Roots of Packed Double-Precision Floating-Point Values</
description>
    </op>
    <op bitmask="0000111101010001" mnemonic="SQRTPS" detail="mem to xmmreg">
        <arg direction="input" type="mem" />
        <arg direction="output" type="xmmreg" />
        <description>Compute Square Roots of Packed Single-Precision Floating-Point Values</
description>
    </op>
    <op bitmask="00001111010101000111" mnemonic="SQRTPS" detail="xmmreg to xmmreg">
        <arg direction="input" type="xmmreg" />
        <arg direction="output" type="xmmreg" />
        <description>Compute Square Roots of Packed Single-Precision Floating-Point Values</
description>
    </op>
    <op bitmask="111100100000111101010001" mnemonic="SQRTSD" detail="mem to xmmreg">
        <arg direction="input" type="mem" />
        <arg direction="output" type="xmmreg" />
        <description>Compute Square Root of Scalar Double-Precision Floating-Point Value</de
scription>
    </op>
    <op bitmask="1111001000001111101010001111" mnemonic="SQRTSD" detail="xmmreg to xmmreg">
        <arg direction="input" type="xmmreg" />
        <arg direction="output" type="xmmreg" />
        <description>Compute Square Root of Scalar Double-Precision Floating-Point Value</de
scription>
    </op>
    <op bitmask="111100110000111101010001" mnemonic="SQRTSS" detail="mem to xmmreg">
        <arg direction="input" type="mem" />
        <arg direction="output" type="xmmreg" />
        <description>Compute Square Root of Scalar Single-Precision Floating-Point Value</de
scription>
    </op>
    <op bitmask="1111001100001111101010001111" mnemonic="SQRTSS" detail="xmmreg to xmmreg">
        <arg direction="input" type="xmmreg" />
        <arg direction="output" type="xmmreg" />
        <description>Compute Square Root of Scalar Single-Precision Floating-Point Value</de
scription>
    </op>
    <op bitmask="11111001" mnemonic="STC">
        <description>Set Carry Flag</description>
```

```
    </op>
    <op bitmask="11111101" mnemonic="STD">
        <description>Set Direction Flag</description>
    </op>
    <op bitmask="11111011" mnemonic="STI">
        <description>Set Interrupt Flag</description>
    </op>
    <op bitmask="00001111101011 10xx011" mnemonic="STMXCSR" detail="MXCSR to mem">
        <arg direction="input" type="MXCSR" />
        <arg direction="output" type="mem" />
        <description>Store MXCSR Register State</description>
    </op>
    <op bitmask="1010101" mnemonic="STOS/STOSB/STOSW/STOSD">
        <description>Store String Data</description>
    </op>
    <op bitmask="0000111100000000011001" mnemonic="STR" detail="to register">
        <arg direction="output" type="reg" />
        <description>Store Task Register</description>
    </op>
    <op bitmask="0000111100000000xx001" mnemonic="STR" detail="to memory">
        <arg direction="output" type="mem" />
        <description>Store Task Register</description>
    </op>
    <op bitmask="0010100" mnemonic="SUB" detail="register to memory">
        <arg direction="input" type="reg" />
        <arg direction="output" type="mem" />
        <description>Integer Subtraction</description>
    </op>
    <op bitmask="0010100x11" mnemonic="SUB" detail="register1 to register2">
        <arg direction="input" type="reg" />
        <arg direction="output" type="reg" />
        <description>Integer Subtraction</description>
    </op>
    <op bitmask="0010101" mnemonic="SUB" detail="memory to register">
        <arg direction="input" type="mem" />
        <arg direction="output" type="reg" />
        <description>Integer Subtraction</description>
    </op>
    <op bitmask="0010101x11" mnemonic="SUB" detail="register2 to register1">
        <arg direction="input" type="reg" />
        <arg direction="output" type="reg" />
        <description>Integer Subtraction</description>
    </op>
    <op bitmask="0010110" mnemonic="SUB" detail="immediate to AL, AX, or EAX">
        <arg direction="input" type="imm" />
        <arg direction="output" type="reg" />
        <description>Integer Subtraction</description>
    </op>
    <op bitmask="100000xx11101" mnemonic="SUB" detail="immediate to register">
        <arg direction="input" type="imm" />
        <arg direction="output" type="reg" />
        <description>Integer Subtraction</description>
    </op>
    <op bitmask="100000xxxx101" mnemonic="SUB" detail="immediate to memory">
        <arg direction="input" type="imm" />
        <arg direction="output" type="mem" />
        <description>Integer Subtraction</description>
    </op>
    <op bitmask="011001100000111101011100" mnemonic="SUBPD" detail="mem to xmmreg">
        <arg direction="input" type="mem" />
        <arg direction="output" type="xmmreg" />
        <description>Subtract Packed Double-Precision Floating-Point Values</description>
    </op>
    <op bitmask="011001100000111101011 10011" mnemonic="SUBPD" detail="xmmreg to xmmreg">
        <arg direction="input" type="xmmreg" />
```

```
        <arg direction="output" type="xmmreg" />
        <description>Subtract Packed Double-Precision Floating-Point Values</description>
    </op>
    <op bitmask="0000111101011100" mnemonic="SUBPS" detail="mem to xmmreg">
        <arg direction="input" type="mem" />
        <arg direction="output" type="xmmreg" />
        <description>Subtract Packed Single-Precision Floating-Point Values</description>
    </op>
    <op bitmask="0000111101011100011" mnemonic="SUBPS" detail="xmmreg to xmmreg">
        <arg direction="input" type="xmmreg" />
        <arg direction="output" type="xmmreg" />
        <description>Subtract Packed Single-Precision Floating-Point Values</description>
    </op>
    <op bitmask="11110010000011110101110100" mnemonic="SUBSD" detail="mem to xmmreg">
        <arg direction="input" type="mem" />
        <arg direction="output" type="xmmreg" />
        <description>Subtract Scalar Double-Precision Floating-Point Values</description>
    </op>
    <op bitmask="11110010000011110101110011" mnemonic="SUBSD" detail="xmmreg to xmmreg">
        <arg direction="input" type="xmmreg" />
        <arg direction="output" type="xmmreg" />
        <description>Subtract Scalar Double-Precision Floating-Point Values</description>
    </op>
    <op bitmask="11110011000011110101110100" mnemonic="SUBSS" detail="mem to xmmreg">
        <arg direction="input" type="mem" />
        <arg direction="output" type="xmmreg" />
        <description>Subtract Scalar Single-Precision Floating-Point Values</description>
    </op>
    <op bitmask="11110011000011110101110011" mnemonic="SUBSS" detail="xmmreg to xmmreg">
        <arg direction="input" type="xmmreg" />
        <arg direction="output" type="xmmreg" />
        <description>Subtract Scalar Single-Precision Floating-Point Values</description>
    </op>
    <op bitmask="0000111100110100" mnemonic="SYSENTER">
        <description>Fast System Call</description>
    </op>
    <op bitmask="0000111100110101" mnemonic="SYSEXIT">
        <description>Fast Return from Fast System Call</description>
    </op>
    <op bitmask="1000010" mnemonic="TEST" detail="memory and register">
        <arg direction="input" type="mem" />
        <arg direction="input" type="reg" />
        <description>Logical Compare</description>
    </op>
    <op bitmask="1000010x11" mnemonic="TEST" detail="register1 and register2">
        <arg direction="input" type="reg" />
        <arg direction="input" type="reg" />
        <description>Logical Compare</description>
    </op>
    <op bitmask="1010100" mnemonic="TEST" detail="immediate and AL, AX, or EAX">
        <arg direction="input" type="imm" />
        <arg direction="input" type="reg" />
        <description>Logical Compare</description>
    </op>
    <op bitmask="1111011x11000" mnemonic="TEST" detail="immediate and register">
        <arg direction="input" type="imm" />
        <arg direction="input" type="reg" />
        <description>Logical Compare</description>
    </op>
    <op bitmask="1111011xxx000" mnemonic="TEST" detail="immediate and memory">
        <arg direction="input" type="imm" />
        <arg direction="input" type="mem" />
        <description>Logical Compare</description>
    </op>
    <op bitmask="01100110000011110010111O" mnemonic="UCOMISD" detail="mem to xmmreg">
```

```
        <arg direction="input" type="mem" />
        <arg direction="input" type="xmmreg" />
        <description>Unordered Compare Scalar Ordered Double-Precision Floating-Point Values
 and Set EFLAGS</description>
    </op>
    <op bitmask="01100110000011110010111011" mnemonic="UCOMISD" detail="xmmreg to xmmreg">
        <arg direction="input" type="xmmreg" />
        <arg direction="input" type="xmmreg" />
        <description>Unordered Compare Scalar Ordered Double-Precision Floating-Point Values
 and Set EFLAGS</description>
    </op>
    <op bitmask="0000111100101110" mnemonic="UCOMISS" detail="mem to xmmreg">
        <arg direction="input" type="mem" />
        <arg direction="input" type="xmmreg" />
        <description>Unordered Compare Scalar Ordered Single-Precision Floating-Point Values
 and Set EFLAGS</description>
    </op>
    <op bitmask="00001111010111011" mnemonic="UCOMISS" detail="xmmreg to xmmreg">
        <arg direction="input" type="xmmreg" />
        <arg direction="input" type="xmmreg" />
        <description>Unordered Compare Scalar Ordered Single-Precision Floating-Point Values
 and Set EFLAGS</description>
    </op>
    <op bitmask="0000111100001011" mnemonic="UD2">
        <description>Undefined instruction</description>
    </op>
    <op bitmask="01100110000011110010101" mnemonic="UNPCKHPD" detail="mem to xmmreg">
        <arg direction="input" type="mem" />
        <arg direction="output" type="xmmreg" />
        <description>Unpack and Interleave High Packed Double-Precision Floating-Point Value
s</description>
    </op>
    <op bitmask="01100110000011110001010111" mnemonic="UNPCKHPD" detail="xmmreg to xmmreg">
        <arg direction="input" type="xmmreg" />
        <arg direction="output" type="xmmreg" />
        <description>Unpack and Interleave High Packed Double-Precision Floating-Point Value
s</description>
    </op>
    <op bitmask="0000111100010101" mnemonic="UNPCKHPS" detail="mem to xmmreg">
        <arg direction="input" type="mem" />
        <arg direction="output" type="xmmreg" />
        <description>Unpack and Interleave High Packed Single-Precision Floating-Point Value
s</description>
    </op>
    <op bitmask="00001111000101011 1" mnemonic="UNPCKHPS" detail="xmmreg to xmmreg">
        <arg direction="input" type="xmmreg" />
        <arg direction="output" type="xmmreg" />
        <description>Unpack and Interleave High Packed Single-Precision Floating-Point Value
s</description>
    </op>
    <op bitmask="01100110000011110010100" mnemonic="UNPCKLPD" detail="mem to xmmreg">
        <arg direction="input" type="mem" />
        <arg direction="output" type="xmmreg" />
        <description>Unpack and Interleave Low Packed Double-Precision Floating-Point Values
</description>
    </op>
    <op bitmask="01100110000011110001010011" mnemonic="UNPCKLPD" detail="xmmreg to xmmreg">
        <arg direction="input" type="xmmreg" />
        <arg direction="output" type="xmmreg" />
        <description>Unpack and Interleave Low Packed Double-Precision Floating-Point Values
</description>
    </op>
    <op bitmask="0000111100010100" mnemonic="UNPCKLPS" detail="mem to xmmreg">
        <arg direction="input" type="mem" />
        <arg direction="output" type="xmmreg" />
```

```
        <description>Unpack and Interleave Low Packed Single-Precision Floating-Point Values
</description>
    </op>
    <op bitmask="000011110001010011" mnemonic="UNPCKLPS" detail="xmmreg to xmmreg">
        <arg direction="input" type="xmmreg" />
        <arg direction="output" type="xmmreg" />
        <description>Unpack and Interleave Low Packed Single-Precision Floating-Point Values
</description>
    </op>
    <op bitmask="0000111100000000011100" mnemonic="VERR" detail="register">
        <arg direction="input" type="reg" />
        <description>Verify a Segment for Reading</description>
    </op>
    <op bitmask="0000111100000000xx100" mnemonic="VERR" detail="memory">
        <arg direction="input" type="mem" />
        <description>Verify a Segment for Reading</description>
    </op>
    <op bitmask="0000111100000000011101" mnemonic="VERW" detail="register">
        <arg direction="input" type="reg" />
        <description>Verify a Segment for Writing</description>
    </op>
    <op bitmask="0000111100000000xx101" mnemonic="VERW" detail="memory">
        <arg direction="input" type="mem" />
        <description>Verify a Segment for Writing</description>
    </op>
    <op bitmask="10011011" mnemonic="WAIT/FWAIT">
        <description>Wait until FPU Ready</description>
    </op>
    <op bitmask="0000111100001001" mnemonic="WBINVD">
        <description>Writeback and Invalidate Data Cache</description>
    </op>
    <op bitmask="0000111100110000" mnemonic="WRMSR">
        <description>Write to Model-Specific Register</description>
    </op>
    <op bitmask="000011111100000" mnemonic="XADD" detail="memory, reg">
        <arg direction="input" type="mem" />
        <arg direction="input" type="reg" />
        <arg direction="output" type="mem" />
        <arg direction="output" type="reg" />
        <description>Exchange and Add</description>
    </op>
    <op bitmask="000011111100000x11" mnemonic="XADD" detail="register1, register2">
        <arg direction="input" type="reg" />
        <arg direction="input" type="reg" />
        <arg direction="output" type="reg" />
        <arg direction="output" type="reg" />
        <description>Exchange and Add</description>
    </op>
    <op bitmask="1000011" mnemonic="XCHG" detail="memory with reg">
        <arg direction="input" type="mem" />
        <arg direction="input" type="reg" />
        <arg direction="output" type="mem" />
        <arg direction="output" type="reg" />
        <description>Exchange Register/Memory with Register</description>
    </op>
    <op bitmask="1000011x11" mnemonic="XCHG" detail="register1 with register2">
        <arg direction="input" type="reg" />
        <arg direction="input" type="reg" />
        <arg direction="output" type="reg" />
        <arg direction="output" type="reg" />
        <description>Exchange Register/Memory with Register</description>
    </op>
    <op bitmask="10010" mnemonic="XCHG" detail="AX or EAX with reg">
        <arg direction="input" type="reg" />
        <arg direction="input" type="reg" />
```

```
        <arg direction="output" type="reg" />
        <arg direction="output" type="reg" />
        <description>Exchange Register/Memory with Register</description>
    </op>
    <op bitmask="11010111" mnemonic="XLAT/XLATB">
        <description>Table Look-up Translation</description>
    </op>
    <op bitmask="0011000" mnemonic="XOR" detail="register to memory">
        <arg direction="input" type="mem" />
        <arg direction="input" type="reg" />
        <arg direction="output" type="mem" />
        <description>Logical Exclusive OR</description>
    </op>
    <op bitmask="0011000x11" mnemonic="XOR" detail="register1 to register2">
        <arg direction="input" type="reg" />
        <arg direction="input" type="reg" />
        <arg direction="output" type="reg" />
        <description>Logical Exclusive OR</description>
    </op>
    <op bitmask="0011001" mnemonic="XOR" detail="memory to register">
        <arg direction="input" type="reg" />
        <arg direction="input" type="mem" />
        <arg direction="output" type="reg" />
        <description>Logical Exclusive OR</description>
    </op>
    <op bitmask="0011001x11" mnemonic="XOR" detail="register2 to register1">
        <arg direction="input" type="reg" />
        <arg direction="input" type="reg" />
        <arg direction="output" type="reg" />
        <description>Logical Exclusive OR</description>
    </op>
    <op bitmask="0011010" mnemonic="XOR" detail="immediate to AL, AX, or EAX">
        <arg direction="input" type="reg" />
        <arg direction="input" type="imm" />
        <arg direction="output" type="reg" />
        <description>Logical Exclusive OR</description>
    </op>
    <op bitmask="100000xx11110" mnemonic="XOR" detail="immediate to register">
        <arg direction="input" type="reg" />
        <arg direction="input" type="imm" />
        <arg direction="output" type="reg" />
        <description>Logical Exclusive OR</description>
    </op>
    <op bitmask="100000xxxx110" mnemonic="XOR" detail="immediate to memory">
        <arg direction="input" type="mem" />
        <arg direction="input" type="imm" />
        <arg direction="output" type="mem" />
        <description>Logical Exclusive OR</description>
    </op>
    <op bitmask="011001100000111101010111" mnemonic="XORPD" detail="mem to xmmreg">
        <arg direction="input" type="xmmreg" />
        <arg direction="input" type="mem" />
        <arg direction="output" type="xmmreg" />
        <description>Bitwise Logical OR of Double-Precision Floating-Point Values</descripti
on>
    </op>
    <op bitmask="0110011000001111010101111111" mnemonic="XORPD" detail="xmmreg to xmmreg">
        <arg direction="input" type="xmmreg" />
        <arg direction="input" type="xmmreg" />
        <arg direction="output" type="xmmreg" />
        <description>Bitwise Logical OR of Double-Precision Floating-Point Values</descripti
on>
    </op>
    <op bitmask="0000111101010111" mnemonic="XORPS" detail="mem to xmmreg">
        <arg direction="input" type="xmmreg" />
```

```
        <arg direction="input" type="mem" />
        <arg direction="output" type="xmmreg" />
        <description>Bitwise Logical XOR of Single-Precision Floating-Point Values</descript
ion>
    </op>
    <op bitmask="0000111110101011111" mnemonic="XORPS" detail="xmmreg to xmmreg">
        <arg direction="input" type="xmmreg" />
        <arg direction="input" type="xmmreg" />
        <arg direction="output" type="xmmreg" />
        <description>Bitwise Logical XOR of Single-Precision Floating-Point Values</descript
ion>
    </op>
</oplist>
```