Intel 80x86 Assembly Language OpCodes

The following table provides a list of x86-Assembler mnemonics, that is not complete. Most of them can be found, for others see at **www.intel.com**

54. **JE/JZ** - Jump Equal / Jump Zero
55. **JG/JNLE** - Jump Greater / Jump Not Less or Equal
56. **JGE/JNL** - Jump Greater or Equal / Jump Not Less
57. **JL/JNGE** - Jump Less / Jump Not Greater or Equal
58. **JLE/JNG** - Jump Less or Equal / Jump Not Greater
59. **JMP** - Unconditional Jump
60. **JNC** - Jump Not Carry
61. **JNE/JNZ** - Jump Not Equal / Jump Not Zero
62. **JNO** - Jump Not Overflow
63. **JNS** - Jump Not Signed
64. **JNP/JPO** - Jump Not Parity / Jump Parity Odd
65. **JO** - Jump on Overflow
66. **JP/JPE** - Jump on Parity / Jump on Parity Even
67. **JS** - Jump Signed
68. **LAHF** - Load Register AH From Flags
69. **LAR** - Load Access Rights
70. **LDS** - Load Pointer Using DS
71. **LEA** - Load Effective Address
72. **LEAVE** - Restore Stack for Procedure Exit
73. **LES** - Load Pointer Using ES
74. **LFS** - Load Pointer Using FS
75. **LGDT** - Load Global Descriptor Table
76. **LIDT** - Load Interrupt Descriptor Table
77. **LGS** - Load Pointer Using GS
78. **LLDT** - Load Local Descriptor Table
79. **LMSW** - Load Machine Status Word
80. **LOCK** - Lock Bus
81. **LODS** - Load String
82. **LOOP** - Decrement CX and Loop if CX Not Zero
83. **LOOPE/LOOPZ** - Loop While Equal / Loop While Zero
84. **LOOPNZ/LOOPNE** - Loop While Not Zero / Loop While Not Equal
85. **LSL** - Load Segment Limit
86. **LSS** - Load Pointer Using SS
87. **LTR** - Load Task Register
88. **MOV** - Move Byte or Word
89. **MOVS** - Move String
90. **MOVSX** - Move with Sign Extend
91. **MOVZX** - Move with Zero Extend
92. **MUL** - Unsigned Multiply
93. **NEG** - Two's Complement Negation
94. **NOP** - No Operation
95. **NOT** - One's Compliment Negation
96. **OR** - Inclusive Logical OR
97. **OUT** - Output Data to Port
98. **OUTS** - Output String to Port
99. **POP** - Pop Word off Stack
100. **POPA/POPAD** - Pop All Registers onto Stack
101. **POPF/POPFD** - Pop Flags off Stack
102. **PUSH** - Push Word onto Stack
103. **PUSHA/PUSHAD** - Push All Registers onto Stack
104. **PUSHF/PUSHFD** - Push Flags onto Stack
105. **RCL** - Rotate Through Carry Left
106. **RCR** - Rotate Through Carry Right
107. **REP** - Repeat String Operation
108. **REPE/REPZ** - Repeat Equal / Repeat Zero
109. **REPNE/REPNZ** - Repeat Not Equal / Repeat Not Zero
110. **RET/RETF** - Return From Procedure
111. **ROL** - Rotate Left
112. **ROR** - Rotate Right
113. **SAHF** - Store AH Register into FLAGS
114. **SAL/SHL** - Shift Arithmetic Left / Shift Logical Left

115. **SAR** - Shift Arithmetic Right
116. **SBB** - Subtract with Borrow
117. **SCAS** - Scan String
118. **SETAE/SETNB** - Set if Above or Equal / Set if Not Below
119. **SETB/SETNAE** - Set if Below / Set if Not Above or Equal
120. **SETBE/SETNA** - Set if Below or Equal / Set if Not Above
121. **SETE/SETZ** - Set if Equal / Set if Zero
122. **SETNE/SETNZ** - Set if Not Equal / Set if Not Zero
123. **SETL/SETNGE** - Set if Less / Set if Not Greater or Equal
124. **SETGE/SETNL** - Set if Greater or Equal / Set if Not Less
125. **SETLE/SETNG** - Set if Less or Equal / Set if Not greater or Equal
126. **SETG/SETNLE** - Set if Greater / Set if Not Less or Equal
127. **SETS** - Set if Signed
128. **SETNS** - Set if Not Signed
129. **SETC** - Set if Carry
130. **SETNC** - Set if Not Carry
131. **SETO** - Set if Overflow
132. **SETNO** - Set if Not Overflow
133. **SETP/SETPE** - Set if Parity / Set if Parity Even
134. **SETNP/SETPO** - Set if No Parity / Set if Parity Odd
135. **SGDT** - Store Global Descriptor Table
136. **SIDT** - Store Interrupt Descriptor Table
137. **SHR** - Shift Logical Right
138. **SHLD/SHRD** - Double Precision Shift
139. **SLDT** - Store Local Descriptor Table
140. **SMSW** - Store Machine Status Word
141. **STC** - Set Carry
142. **STD** - Set Direction Flag
143. **STI** - Set Interrupt Flag
144. **STOS** - Store String
145. **STR** - Store Task Register
146. **SUB** - Subtract
147. **TEST** - Test For Bit Pattern
148. **VERR** - Verify Read
149. **VERW** - Verify Write
150. **WAIT/FWAIT** - Event Wait
151. **WBINVD** - Write-Back and Invalidate Cache
152. **XCHG** - Exchange
153. **XLAT/XLATB** - Translate
154. **XOR** - Exclusive OR

Notations and Format used in this Document

Notation

mnemonics
  Instruction syntax

op
  Instruction OpCode

xx
  Additional Code bytes

s
  Sign Bit

    E: Sign-extended 8-bit immediate data
    N: Non

w
  Word/byte Bit

W: 16-bit operanrs
B: 8-bit operanrs

len
Instruction length
flags

`--------c` - Carry flag
`-------p-` - Parity flag
`------a--` - Auxiliary flag
`-----z---` - Zero flag
`----s----` - Sign flag
`---t-----` - Trap flag
`--i------` - Interrupt flag
`-d-------` - Direction flag
`o--------` - Overflow flag

`mr`
Addressing mode Byte = MODRM(mod-reg-r/m)
`/0~7`
2nd or 3rd Opcode (MODRM bits 5,4,3 from reg field)
`d0 d1`
Displacement [Low-byte High-byte]
`i0 i1`
Immediate word value
`o0 o1`
Offset value
`s0 s1`
Segment value

`r0`
Relative Short Displacement to label 'sl' (-128/+127 bytes)
`r0 r1`
Relative Long Displacement to label 'll' (-32768/+32767 bytes)

Mnemonic Notation

`mb`	memory byte	`rb`	register byte	`rmb`	register or memory byte
`mw`	memory word	`rw`	register word	`rmw`	register or memory word
`md`	memory double word	`rd`	register double word		
`mq`	memory quad word				
`sl`	short label	`ib`	immediate byte	`mwr`	memory word real
`ll`	long label	`iw`	immediate word	`mdr`	memory double word real
				`mqr`	memory quad word real
`np`	near pointer			`mtr`	memory ten byte real
`fp`	far pointer				
`cr`		`dr`		`tr`	

control register debug register test register

Instruction General Format

PreFix	OpCode	modRM	Disp	Imm

AAA - Ascii Adjust for Addition

mnemonics	op xx xx xx xx xx	sw	len	flags
AAA	37		1	------a-

Usage
```
AAA
```
Modifies flags

AF CF (OF,PF,SF,ZF undefined)

Changes contents of AL to valid unpacked decimal.
The high order nibble is zeroed.

AAD - Ascii Adjust for Division

mnemonics	op xx xx xx xx xx	sw	len	flags
AAD	D5 0A		2	----sz-p

Usage
```
AAD
```
Modifies flags

SF ZF PF (AF,CF,OF undefined)

Used before dividing unpacked decimal numbers.
Multiplies AH by 10 and the adds result into AL. Sets AH to zero.
This instruction is also known to have an undocumented behavior.

AAM - Ascii Adjust for Multiplication

mnemonics	op xx xx xx xx xx	sw	len	flags
AAM	D4 0A		2	----sz-p

Usage
```
AAM
```
Modifies flags

PF SF ZF (AF,CF,OF undefined)

Used after multiplication of two unpacked decimal numbers, this instruction adjusts an unpacked decimal number. The high order nibble of each byte must be zeroed before using this instruction. This instruction is also known to have an undocumented behavior.

AAS - Ascii Adjust for Subtraction

mnemonics	op xx xx xx xx xx	sw	len	flags
AAS	3F		1	------a-

Usage
```
AAS
```

Modifies flags
    AF CF (OF,PF,SF,ZF undefined)

Corrects result of a previous unpacked decimal subtraction in AL.
High order nibble is zeroed.

ADC - Add With Carry

mnemonics	op xx xx xx xx xx	sw	len	flags
ADC AL,ib	14 i0	B	2	o---szap
ADC AX,iw	15 i0 i1	W	3	o---szap
ADC rb,rmb	12 mr d0 d1	B	2~4	o---szap
ADC rw,rmw	13 mr d0 d1	W	2~4	o---szap
ADC rmb,ib	80 /2 d0 d1 i0	NB	3~5	o---szap
ADC rmw,iw	81 /2 d0 d1 i0 i1	NW	4~6	o---szap
ADC rmw,ib	83 /2 d0 d1 i0	EW	3~5	o---szap
ADC rmb,rb	10 mr d0 d1	B	2~4	o---szap
ADC rmw,rw	11 mr d0 d1	W	2~4	o---szap

Usage
```
    ADC     dest,src
```
Modifies flags
    AF CF OF SF PF ZF

Sums two binary operands placing the result in the destination.
If CF is set, a 1 is added to the destination.

ADD - Arithmetic Addition

mnemonics	op xx xx xx xx xx	sw	len	flags
ADD AL,ib	04 i0	B	2	o---szap
ADD AX,iw	05 i0 i1	W	3	o---szap
ADD rb,rmb	02 mr d0 d1	B	2~4	o---szap
ADD rw,rmw	03 mr d0 d1	W	2~4	o---szap
ADD rmb,ib	80 /0 d0 d1 i0	NB	3~5	o---szap
ADD rmw,iw	81 /0 d0 d1 i0 i1	NW	4~6	o---szap
ADD rmw,ib	83 /0 d0 d1 i0	EW	3~5	o---szap
ADD rmb,rb	00 mr d0 d1	B	2~4	o---szap
ADD rmw,rw	01 mr d0 d1	W	2~4	o---szap

Usage
```
    ADD     dest,src
```
Modifies flags
    AF CF OF PF SF ZF

Adds "src" to "dest" and replacing the original contents of "dest".
Both operands are binary.

AND - Logical And

mnemonics	op xx xx xx xx xx	sw	len	flags
AND AL,ib	24 i0	B	2	0---sz-p
AND AX,iw	25 i0 i1	W	3	0---sz-p

AND rb,rmb	22 mr d0 d1	B	2~4	0---sz-p
AND rw,rmw	23 mr d0 d1	W	2~4	0---sz-p
AND rmb,ib	80 /4 d0 d1 i0	NB	3~5	0---sz-p
AND rmw,iw	81 /4 d0 d1 i0 i1	NW	4~6	0---sz-p
AND rmw,ib	83 /4 d0 d1 i0	EW	3~5	0---sz-p
AND rmb,rb	20 mr d0 d1	B	2~4	0---sz-p
AND rmw,rw	21 mr d0 d1	W	2~4	0---sz-p

Usage

```
        AND     dest,src
```

Modifies flags

CF OF PF SF ZF (AF undefined)

Performs a logical AND of the two operands replacing the destination with the result.

ARPL - Adjusted Requested Privilege Level of Selector (286+ protected mode)

mnemonics	op xx xx xx xx xx	sw	len	flags
ARPL rmw,rw [286]	63 mr d0 d1		2~4	-----z--

Usage

```
        ARPL    dest,src
```

Modifies flags

ZF

Compares the RPL bits of "dest" against "src". If the RPL bits
of "dest" are less than "src", the destination RPL bits are set
equal to the source RPL bits and the Zero Flag is set. Otherwise
the Zero Flag is cleared.

BOUND - Array Index Bound Check (80188+)

mnemonics	op xx xx xx xx xx	sw	len	flags
BOUND rw,rmw [186]	62 mr d0 d1		2~4	--------

Usage

```
        BOUND   src,limit
```

Modifies flags

None

Array index in source register is checked against upper and lower
bounds in memory source. The first word located at "limit" is
the lower boundary and the word at "limit+2" is the upper array bound.
Interrupt 5 occurs if the source value is less than or higher than
the source.

BSF - Bit Scan Forward (386+ only)

mnemonics	op xx xx xx xx xx	sw	len	flags
BSF rw,rmw [386]	0F BC mr d0 d1		3~5	-----z--

Usage

```
        BSF     dest,src
```

Modifies flags

ZF

Scans source operand for first bit set. Sets ZF if a bit is found set and loads the destination with an index to first set bit. Clears ZF is no bits are found set. BSF scans forward across bit pattern (0-n) while BSR scans in reverse (n-0).

BSR - Bit Scan Reverse (386+ only)

mnemonics	op xx xx xx xx xx	sw	len	flags
BSR rw,rmw [386]	0F BD mr d0 d1		3~5	-----z--

Usage
```
     BSR     dest,src
```
Modifies flags
ZF

Scans source operand for first bit set. Sets ZF if a bit is found set and loads the destination with an index to first set bit. Clears ZF is no bits are found set. BSF scans forward across bit pattern (0-n) while BSR scans in reverse (n-0).

BSWAP - Byte Swap (486+ only)

mnemonics	op xx xx xx xx xx	sw	len	flags
BSWAP eax [486]	0F C8		2	--------
BSWAP ecx [486]	0F C9		2	--------
BSWAP edx [486]	0F CA		2	--------
BSWAP ebx [486]	0F CB		2	--------
BSWAP esp [486]	0F CC		2	--------
BSWAP ebp [486]	0F CD		2	--------
BSWAP esi [486]	0F CE		2	--------
BSWAP edi [486]	0F CF		2	--------

Usage
```
     BSWAP   reg32
```
Modifies flags
none

Changes the byte order of a 32 bit register from big endian to little endian or vice versa. Result left in destination register is undefined if the operand is a 16 bit register.

BT - Bit Test (386+ only)

mnemonics	op xx xx xx xx xx	sw	len	flags
BT rmw,ib [386]	0F BA /4 d0 d1 i0		4~6	--------
BT rmw,rw [386]	0F A3 mr d0 d1		3~5	--------

Usage
```
     BT      dest,src
```
Modifies flags
CF

The destination bit indexed by the source value is copied into the Carry Flag.

BTC - Bit Test with Compliment (386+ only)

mnemonics	op xx xx xx xx xx	sw	len	flags
BTC rmw,ib [386]	0F BA /7 d0 d1 i0		4~6	--------
BTC rmw,rw [386]	0F BB mr d0 d1		3~5	--------

Usage
`BTC dest,src`
Modifies flags
CF

The destination bit indexed by the source value is copied into the Carry Flag after being complimented (inverted).

BTR - Bit Test with Reset (386+ only)

mnemonics	op xx xx xx xx xx	sw	len	flags
BTR rmw,ib [386]	0F BA /6 d0 d1 i0		4~6	--------
BTR rmw,rw [386]	0F B3 mr d0 d1		3~5	--------

Usage
`BTR dest,src`
Modifies flags
CF

The destination bit indexed by the source value is copied into the Carry Flag and then cleared in the destination.

BTS - Bit Test and Set (386+ only)

mnemonics	op xx xx xx xx xx	sw	len	flags
BTS rmw,ib [386]	0F BA /5 d0 d1 i0		4~6	--------
BTS rmw,rw [386]	0F AB mr d0 d1		3~5	--------

Usage
`BTS dest,src`
Modifies flags
CF

The destination bit indexed by the source value is copied into the Carry Flag and then set in the destination.

CALL - Procedure Call

mnemonics	op xx xx xx xx xx	sw	len	flags
CALL np	E8 o0 o1		3	--------
CALL rw	FF /2 d0 d1	W	2~4	--------
CALL DWORD PTR[rw]	FF /3 d0 d1	W	2~4	--------
CALL FAR PTR fp	9A o0 o1 sl sh		5	--------

Usage
`CALL destination`
Modifies flags
None

Pushes Instruction Pointer (and Code Segment for far calls) onto stack and loads Instruction Pointer with the address of proc-name.

Code continues with execution at CS:IP.

CBW - Convert Byte to Word

mnemonics	op xx xx xx xx xx	sw	len	flags
CBW	98		1	--------

Usage
`CBW`
Modifies flags
None

Converts byte in AL to word Value in AX by extending sign of AL throughout register AH.

CDQ - Convert Double to Quad (386+ only)

mnemonics	op xx xx xx xx xx	sw	len	flags
CDQ [32bit]	66│ 99		1+1	--------

Usage
`CDQ`
Modifies flags
None

Converts signed DWORD in EAX to a signed quad word in EDX:EAX by extending the high order bit of EAX throughout EDX

CLC - Clear Carry

mnemonics	op xx xx xx xx xx	sw	len	flags
CLC	F8		1	--------

Usage
`CLC`
Modifies flags
CF

Clears the Carry Flag.

CLD - Clear Direction Flag

mnemonics	op xx xx xx xx xx	sw	len	flags
CLD	FC		1	-0------

Usage
`CLD`
Modifies flags
DF

Clears the Direction Flag causing string instructions to increment the SI and DI index registers.

CLI - Clear Interrupt Flag (Disable Interrupts)

mnemonics	op xx xx xx xx xx	sw	len	flags
CLI	FA		1	--0-----

Usage
`CLI`
Modifies flags
IF

Disables the maskable hardware interrupts by clearing the Interrupt flag. NMI's and software interrupts are not inhibited.

CLTS - Clear Task Switched Flag (286+ privileged)

mnemonics	op xx xx xx xx xx	sw	len	flags
CLTS [286]	0F 06		2	--------

Usage
`CLTS`
Modifies flags
None

Clears the Task Switched Flag in the Machine Status Register. This is a privileged operation and is generally used only by operating system code.

CMC - Complement Carry Flag

mnemonics	op xx xx xx xx xx	sw	len	flags
CMC	F5		1	--------

Usage
`CMC`
Modifies flags
CF

Toggles (inverts) the Carry Flag

CMP - Compare

mnemonics	op xx xx xx xx xx	sw	len	flags
CMP AL,ib	3C i0	B	2	o---szap
CMP AX,iw	3D i0 i1	W	3	o---szap
CMP rb,rmb	3A mr d0 d1	B	2~4	o---szap
CMP rw,rmw	3B mr d0 d1	W	2~4	o---szap
CMP rmb,ib	80 /7 d0 d1 i0	NB	3~5	o---szap
CMP rmw,iw	81 /7 d0 d1 i0 i1	NW	4~6	o---szap
CMP rmw,ib	83 /7 d0 d1 i0	EW	3~5	o---szap
CMP rmb,rb	38 mr d0 d1	B	2~4	o---szap
CMP rmw,rw	39 mr d0 d1	W	2~4	o---szap

Usage
`CMP     dest,src`
Modifies flags
AF CF OF PF SF ZF

Subtracts source from destination and updates the flags but does not save result. Flags can subsequently be checked for conditions.

CMPS - Compare String (Byte, Word or Doubleword)

mnemonics	op xx xx xx xx xx	sw	len	flags
CMPSB	A6	B	1	od--szap
CMPSW	A7	W	1	od--szap
CMPSD [32bit]	66│ A7	D	1+1	od--szap

Usage

```
CMPS    dest,src
CMPSB
CMPSW
CMPSD   (386+ only)
```

Modifies flags

AF CF OF PF SF ZF

Subtracts destination value from source without saving results. Updates flags based on the subtraction and the index registers (E)SI and (E)DI are incremented or decremented depending on the state of the Direction Flag. CMPSB inc/decrements the index registers by 1, CMPSW inc/decrements by 2, while CMPSD increments or decrements by 4. The REP prefixes can be used to process entire data items.

CMPXCHG/CMPXCHG8B - Compare and Exchange

mnemonics	op xx xx xx xx xx	sw	len	flags
CMPXCHG rmb,rb [486]	0F A6 mr d0 d1	B	3~5	o---szap
CMPXCHG rmw,rw [486]	0F A7 mr d0 d1	W	3~5	o---szap
CMPXCHG rmb,rb [486]	0F B0 mr d0 d1	B	3~5	o---szap
CMPXCHG rmw,rw [486]	0F B1 mr d0 d1	W	3~5	o---szap
CMPXCHG8B rmq,rd [P5]	0F C7 mr d0 d1		3~5	-----z--

Usage

```
CMPXCHG dest,src  (486+)
CMPXCHG8B dest, src (P5+)
```

Modifies flags

AF CF OF PF SF ZF

Compares the accumulator (8-32 bits) with "dest". If equal the "dest" is loaded with "src", otherwise the accumulator is loaded with "dest".

CWD - Convert Word to Doubleword

mnemonics	op xx xx xx xx xx	sw	len	flags
CWD	99		1	--------

Usage

```
CWD
```

Modifies flags

None

Extends sign of word in register AX throughout register DX forming a doubleword quantity in DX:AX.

CWDE - Convert Word to Extended Doubleword (386+ only)

mnemonics	op xx xx xx xx xx	sw	len	flags
CWDE [32bit]	66| 98		1+1	--------

Usage

```
    CWDE
```

Modifies flags

None

Converts a signed word in AX to a signed doubleword in EAX by extending the sign bit of AX throughout EAX.

DAA - Decimal Adjust for Addition

mnemonics	op xx xx xx xx xx	sw	len	flags
DAA	27		1	----szap

Usage

```
    DAA
```

Modifies flags

AF CF PF SF ZF (OF undefined)

Corrects result (in AL) of a previous BCD addition operation. Contents of AL are changed to a pair of packed decimal digits.

DAS - Decimal Adjust for Subtraction

mnemonics	op xx xx xx xx xx	sw	len	flags
DAS	2F		1	----szap

Usage

```
    DAS
```

Modifies flags

AF CF PF SF ZF (OF undefined)

Corrects result (in AL) of a previous BCD subtraction operation. Contents of AL are changed to a pair of packed decimal digits.

DEC - Decrement

mnemonics	op xx xx xx xx xx	sw	len	flags
DEC AX	48		1	o---szap
DEC BP	4C		1	o---szap
DEC BX	4A		1	o---szap
DEC CX	49		1	o---szap
DEC DI	4F		1	o---szap
DEC DX	49		1	o---szap
DEC rmb	FE /1 d0 d1		2~4	o---szap
DEC rmw	FF /1 d0 d1		2~4	o---szap
DEC SI	4D		1	o---szap
DEC SP	4B		1	o---szap

Usage

```
    DEC     dest
```

Modifies flags

AF OF PF SF ZF

Unsigned binary subtraction of one from the destination.

DIV - Divide

mnemonics	op xx xx xx xx xx	sw	len	flags
DIV rmb	F6 /6 d0 d1	B	2~4	o---szap
DIV rmw	F7 /6 d0 d1	W	2~4	o---szap

Usage

```
    DIV     src
```

Modifies flags

(AF,CF,OF,PF,SF,ZF undefined)

Unsigned binary division of accumulator by source. If the source divisor is a byte value then AX is divided by "src" and the quotient is placed in AL and the remainder in AH. If source operand is a word value, then DX:AX is divided by "src" and the quotient is stored in AX and the remainder in DX.

ENTER - Make Stack Frame (80188+)

mnemonics	op xx xx xx xx xx	sw	len	flags
ENTER iw,ib [186]	C8 i0 i1 i0		4	--------

Usage

```
    ENTER   locals,level
```

Modifies flags

None

Modifies stack for entry to procedure for high level language. Operand "locals" specifies the amount of storage to be allocated on the stack. "Level" specifies the nesting level of the routine. Paired with the LEAVE instruction, this is an efficient method of entry and exit to procedures.

A description of the floating point instructions is not available at yet.

The following table has been provided for op-codes:

mnemonics	op xx xx xx xx xx	sw	len	flags
F2XM1	D9 F0		2	--------
FABS	D9 E1		2	--------
FADD	DE C1		2	--------
FADD mdr	D8 /0 d0 d1	D	2~4	--------
FADD mqr	DC /0 d0 d1	Q	2~4	--------
FADD st(i),st	DC C0+i		2	--------
FADD st,st(i)	D8 C0+i		2	--------
FADDP st(i),st	DE C0+i		2	--------
FBLD mtr	DF /4 d0 d1		2~4	--------
FBSTP mtr	DF /6 d0 d1		2~4	--------
FCHS	D9 E0		2	--------
FCLEX	9B DB E2		3	--------
FCOM	D8 D1		2	-----z-p

FCOM mdr	D8 /2 d0 d1	D	2~4	-----z-p
FCOM mqr	DC /2 d0 d1	Q	2~4	-----z-p
FCOM st(i)	D8 D0+i		2	-----z-p
FCOMP	D8 D9		2	-----z-p
FCOMP mdr	D8 /3 d0 d1	D	2~4	-----z-p
FCOMP mqr	DC /3 d0 d1	Q	2~4	-----z-p
FCOMP st(i)	D8 D8+i		2	-----z-p
FCOMPP	DE D9		2	-----z-p
FCOS [387]	D9 FF		2	--------
FDECSTP	D9 F6		2	--------
FDISI	9B DB E1		3	--------
FDIV mdr	D8 /6 d0 d1	D	2~4	--------
FDIV mqr	DC /6 d0 d1	Q	2~4	--------
FDIV st(i),st	DC F8+i		2	--------
FDIV st,st(i)	DC F0+i		2	--------
FDIVP	DE F9		2	--------
FDIVP st(i),st	DE F8+i		2	--------
FDIVR mdr	D8 /7 d0 d1	D	2~4	--------
FDIVR mqr	DC /7 d0 d1	Q	2~4	--------
FDIVR st(i),st	DC F0+i		2	--------
FDIVR st,st(i)	DC F8+i		2	--------
FDIVRP	DE F1		2	--------
FDIVRP st(i),st	DE F0+i		2	--------
FENI	9B DB E0		3	--------
FFREE st(i)	DD C0+i		2	--------
FIADD mw	DE /0 d0 d1	W	2~4	--------
FIADD md	DA /0 d0 d1	D	2~4	--------
FICOM mdr	DE /2 d0 d1	D	2~4	-----z-p
FICOM mqr	DA /2 d0 d1	Q	2~4	-----z-p
FICOMP md	DE /3 d0 d1	D	2~4	-----z-p
FICOMP mq	DA /3 d0 d1	Q	2~4	-----z-p
FIDIV mw	DE /6 d0 d1	W	2~4	--------
FIDIV md	DA /6 d0 d1	D	2~4	--------
FIDIVR mw	DE /7 d0 d1	W	2~4	--------
FIDIVR md	DA /7 d0 d1	D	2~4	--------
FILD mw	DF /0 d0 d1	W	2~4	--------
FILD md	DB /0 d0 d1	D	2~4	--------
FILD mq	DF /5 d0 d1	Q	2~4	--------
FIMUL mw	DE /1 d0 d1	W	2~4	--------
FIMUL md	DA /1 d0 d1	D	2~4	--------
FINCSTP	D9 F7		2	--------
FINIT	9B DB E3		3	--------
FIST mw	DF /2 d0 d1	W	2~4	--------
FIST md	DB /2 d0 d1	D	2~4	--------
FISTP mw	DF /3 d0 d1	W	2~4	--------
FISTP md	DB /3 d0 d1	D	2~4	--------
FISTP mq	DF /7 d0 d1	Q	2~4	--------
FISUB mw	DE /4 d0 d1	W	2~4	--------

FISUB md	DA /4 d0 d1	D	2~4	--------
FISUBR mw	DE /5 d0 d1	W	2~4	--------
FISUBR md	DA /5 d0 d1	D	2~4	--------
FLD mdr	D9 /0 d0 d1	D	2~4	--------
FLD mqr	DD /0 d0 d1	Q	2~4	--------
FLD mtr	DB /5 d0 d1	T	2~4	--------
FLD st(i)	D9 C0+i		2	--------
FLD1	D9 E8		2	--------
FLDCW mw	D9 /5 d0 d1	W	2~4	--------
FLDENV m14	D9 /4 d0 d1		2~4	--------
FLDL2E	D9 EA		2	--------
FLDL2T	D9 E9		2	--------
FLDLG2	D9 EC		2	--------
FLDLN2	D9 ED		2	--------
FLDPI	D9 EB		2	--------
FLDZ	D9 EE		2	--------
FMUL	DE C9		2	--------
FMUL mdr	D8 /1 d0 d1	D	2~4	--------
FMUL mqr	DC /1 d0 d1	Q	2~4	--------
FMUL st(i),st	DC C8+i		2	--------
FMUL st,st(i)	D8 C8+i		2	--------
FMULP st(i),st	DE C8+i		2	--------
FNCLEX	DB E2		2	--------
FNDISI	DB E1		2	--------
FNENI	DB E0		2	--------
FNINIT	DB E3		2	--------
FNOP	D9 D0		2	--------
FNSAVE m94	DD /6 d0 d1		2~4	--------
FNSTCW mw	D9 /7 d0 d1	W	2~4	--------
FNSTENV m14	D9 /6 d0 d1		2~4	--------
FNSTSW ax	DF E0		2	--------
FNSTSW mw	DD /7 d0 d1	W	2~4	--------
FPATAN	D9 F3		2	--------
FPREM	D9 F8		2	--------
FPREM1 [387]	D9 F5		2	--------
FPTAN	D9 F2		2	--------
FRNDINT	D9 FC		2	--------
FRSTOR m94	DD /4 d0 d1		2~4	--------
FSAVE m94	9B DD /6 d0 d1		3~5	--------
FSCALE	D9 FD		2	--------
FSETPM	DB E4		2	--------
FSIN [387]	D9 FE		2	--------
FSINCOS [387]	D9 FB		2	--------
FSQRT	D9 FA		2	--------
FST mdr	D9 /2 d0 d1	D	2~4	--------
FST mqr	DD /2 d0 d1	Q	2~4	--------
FST st(i)	DD D0+i		2	--------
FSTCW mw	9B D9 /7 d0 d1	W	3~5	--------

FSTENV m14	9B D9 /6 d0 d1		3~5	--------
FSTP mdr	D9 /3 d0 d1	D	2~4	--------
FSTP mqr	DD /3 d0 d1	Q	2~4	--------
FSTP mtr	DB /7 d0 d1	T	2~4	--------
FSTP st(i)	DD D8+i		2	--------
FSTSW ax	9B DF E0		3	--------
FSTSW mw	9B DD /7 d0 d1	W	3~5	--------
FSUB mdr	D8 /4 d0 d1	D	2~4	--------
FSUB mqr	DC /4 d0 d1	Q	2~4	--------
FSUB st(i),st	DC E8+i		2	--------
FSUB st,st(i)	D8 E0+i		2	--------
FSUBP	DE E9		2	--------
FSUBP st(i),st	DE E8+i		2	--------
FSUBR	DE E1		2	--------
FSUBR mdr	D8 /5 d0 d1	D	2~4	--------
FSUBR mqr	DC /5 d0 d1	Q	2~4	--------
FSUBR st(i),st	DC E0+i		2	--------
FSUBR st,st(i)	D8 E8+i		2	--------
FSUBRP st(i),st	DE E0+i		2	--------
FTST	D9 E4		2	--------
FUCOM [387]	DD E1		2	-----z-p
FUCOM st(i) [387]	DD E0+i		2	-----z-p
FUCOMP st(i) [387]	DD E8+i		2	-----z-p
FUCOMPP [387]	DA E9		2	-----z-p
FXAM	D9 E5		2	--------
FXCH	D9 C9		2	--------
FXCH st(i)	D9 C8+i		2	--------
FXTRACT	D9 F4		2	--------
FYL2X	D9 F1		2	--------
FYL2XP1	D9 F9		2	--------

ESC - Escape

mnemonics	op xx xx xx xx xx	sw	len	flags
ESC	?		2	--------

Usage

```
ESC     immed,src
```

Modifies flags

None

Provides access to the data bus for other resident processors.
The CPU treats it as a NOP but places memory operand on bus.

HLT - Halt CPU

mnemonics	op xx xx xx xx xx	sw	len	flags
HLT	F4		1	--------

Usage

```
      HLT
```

Modifies flags

None

Halts CPU until RESET line is activated, NMI or maskable interrupt received. The CPU becomes dormant but retains the current CS:IP for later restart.

IDIV - Signed Integer Division

mnemonics	op xx xx xx xx xx	sw	len	flags
IDIV rmb	F6 /7 d0 d1	B	2~4	o---szap
IDIV rmw	F7 /7 d0 d1	W	2~4	o---szap

Usage

```
      IDIV   src
```

Modifies flags

(AF,CF,OF,PF,SF,ZF undefined)

Signed binary division of accumulator by source. If source is a byte value, AX is divided by "src" and the quotient is stored in AL and the remainder in AH. If source is a word value, DX:AX is divided by "src", and the quotient is stored in AL and the remainder in DX.

IMUL - Signed Multiply

mnemonics	op xx xx xx xx xx	sw	len	flags
IMUL rb,rmb [386]	0F AF mr d0 d1	B	3~5	o---szap
IMUL rd,ib	6B mr i0	W	3	o---szap
IMUL rd,id	69 mr i0 i1 i2 i3	W	6	o---szap
IMUL rd,rmd,ib	6B mr d0 d1 i0	W	3~5	o---szap
IMUL rd,rmd,id	69 mr d0 d1 i0~i3	W	6~8	o---szap
IMUL rmb	F6 /5 d0 d1	B	2~4	o---szap
IMUL rmw	F7 /5 d0 d1	W	2~4	o---szap
IMUL rw,ib	6B mr i0	B	3	o---szap
IMUL rw,iw	69 mr i0 i1	B	4	o---szap
IMUL rw,rmw [386]	0F AF mr d0 d1	W	3~5	o---szap
IMUL rw,rmw,ib	6B mr d0 d1 i0	B	3~5	o---szap
IMUL rw,rmw,iw	69 mr d0 d1 i0 i1	B	4~6	o---szap

Usage

```
      IMUL    src
      IMUL    src,immed  (286+ only)
      IMUL    dest,src,immed8  (286+ only)
      IMUL    dest,src  (386+ only)
```

Modifies flags

CF OF (AF,PF,SF,ZF undefined)

Signed multiplication of accumulator by "src" with result placed in the accumulator. If the source operand is a byte value, it is multiplied by AL and the result stored in AX. If the source operand is a word value it is multiplied by AX and the result is stored in DX:AX. Other variations of this instruction allow specification of source and destination registers as well as a third immediate factor.

IN - Input Byte or Word From Port

mnemonics	op xx xx xx xx xx	sw	len	flags
IN AL,ib	E4 i0	B	2	--------
IN AL,DX	EC	B	1	--------
IN AX,ib	E5 i0	W	2	--------
IN AX,DX	ED	W	1	--------

Usage
```
    IN      accum,port
```
Modifies flags
    None

A byte, word or dword is read from "port" and placed in AL, AX or EAX respectively. If the port number is in the range of 0-255 it can be specified as an immediate, otherwise the port number must be specified in DX. Valid port ranges on the PC are 0-1024, though values through 65535 may be specified and recognized by third party vendors and PS/2's.

INC - Increment

mnemonics	op xx xx xx xx xx	sw	len	flags
INC AX	40		1	o---szap
INC CX	41		1	o---szap
INC DX	42		1	o---szap
INC BX	43		1	o---szap
INC SP	44		1	o---szap
INC BP	45		1	o---szap
INC SI	46		1	o---szap
INC DI	47		1	o---szap
INC rmb	FE /0 d0 d1		2~4	o---szap
INC rmw	FF /0 d0 d1		2~4	o---szap

Usage
```
    INC     dest
```
Modifies flags
    AF OF PF SF ZF

Adds one to destination unsigned binary operand.

INS - Input String from Port (80188+)

mnemonics	op xx xx xx xx xx	sw	len	flags
INSB [186]	6C	B	1	--------
INSW [186]	6D	W	1	--------
INSD [32bit]	66| 6D	D	1+1	--------

Usage
```
    INS     dest,port
    INSB
```

```
      INSW
      INSD  (386+ only)
```

Modifies flags

None

Loads data from port to the destination ES:(E)DI (even if a destination operand is supplied). (E)DI is adjusted by the size of the operand and increased if the Direction Flag is cleared and decreased if the Direction Flag is set. For INSB, INSW, INSD no operands are allowed and the size is determined by the mnemonic.

INT - Interrupt

mnemonics	op xx xx xx xx xx	sw	len	flags
INT 3	CC		1	--00----
INT ib	CD i0		2	--00----

Usage

```
      INT     num
```

Modifies flags

TF IF

Initiates a software interrupt by pushing the flags, clearing the Trap and Interrupt Flags, pushing CS followed by IP and loading CS:IP with the value found in the interrupt vector table. Execution then begins at the location addressed by the new CS:IP

INTO - Interrupt on Overflow

mnemonics	op xx xx xx xx xx	sw	len	flags
INTO	CE		1	--00----

Usage

```
      INTO
```

Modifies flags

IF TF

If the Overflow Flag is set this instruction generates an INT 4 which causes the code addressed by 0000:0010 to be executed.

INVD - Invalidate Cache (486+ only)

mnemonics	op xx xx xx xx xx	sw	len	flags
INVD [486]	0F 08		2	--------

Usage

```
      INVD
```

Modifies flags

none

Flushes CPU internal cache. Issues special function bus cycle which indicates to flush external caches. Data in write-back external caches is lost.

INVLPG - Invalidate Translation Look-Aside Buffer Entry (486+ only)

mnemonics	op xx xx xx xx xx	sw	len	flags
INVLPG m [486]	0F 01 /7		3	--------

Usage
```
INVLPG
```
Modifies flags
none

Invalidates a single page table entry in the Translation Look-Aside Buffer. Intel warns that this instruction may be implemented differently on future processors.

IRET/IRETD - Interrupt Return

mnemonics	op xx xx xx xx xx	sw	len	flags
IRET	CF		1	oditszap
IRETD [32bit]	66| CF		1+1	oditszap

Usage
```
IRET
IRETD  (386+ only)
```
Modifies flags
AF CF DF IF PF SF TF ZF

Returns control to point of interruption by popping IP, CS and then the Flags from the stack and continues execution at this location. CPU exception interrupts will return to the instruction that cause the exception because the CS:IP placed on the stack during the interrupt is the address of the offending instruction.

JA/JNBE - Jump Above / Jump Not Below or Equal

mnemonics	op xx xx xx xx xx	sw	len	flags
JA ll [386]	0F 87 r0 r1		4	--------
JA sl	77 r0		2	--------

Usage
```
JA      label
JNBE    label
```
Modifies flags
None

Causes execution to branch to "label" if the Carry Flag and Zero Flag are both clear. Unsigned comparison.

*) JA/JNBE are different mnemonics for the same instruction

JAE/JNB - Jump Above or Equal / Jump on Not Below

mnemonics	op xx xx xx xx xx	sw	len	flags
JAE ll [386]	0F 83 r0 r1		4	--------
JAE sl	73 r0		2	--------

Usage
```
JAE     label
```

```
      JNB     label
```

Modifies flags

None

Causes execution to branch to "label" if the Carry Flag is clear.
Functionally similar to JNC. Unsigned comparison.

*) JAE/JNB are different mnemonics for the same instruction

JB/JNAE - Jump Below / Jump Not Above or Equal

mnemonics	op xx xx xx xx xx	sw	len	flags
JB ll [386]	0F 82 r0 r1		4	--------
JB sl	72 r0		2	--------

Usage

```
      JB      label
      JNAE    label
```

Modifies flags

None

Causes execution to branch to "label" if the Carry Flag is set.
Functionally similar to JC. Unsigned comparison.

*) JB/JNAE are different mnemonics for the same instruction

JBE/JNA - Jump Below or Equal / Jump Not Above

mnemonics	op xx xx xx xx xx	sw	len	flags
JBE ll [386]	0F 86 r0 r1		4	--------
JBE sl	76 r0		2	--------

Usage

```
      JBE     label
      JNA     label
```

Modifies flags

None

Causes execution to branch to "label" if the Carry Flag or
the Zero Flag is set. Unsigned comparison.

*) JBE/JNA are different mnemonics for the same instruction

JC - Jump on Carry

mnemonics	op xx xx xx xx xx	sw	len	flags
JC ll [386]	0F 82 r0 r1		4	--------
JC sl	72 r0		2	--------

Usage

```
      JC      label
```

Modifies flags

None

Causes execution to branch to "label" if the Carry Flag is set.
Functionally similar to JB and JNAE. Unsigned comparison.

JCXZ/JECXZ - Jump if Register (E)CX is Zero

mnemonics	op xx xx xx xx xx	sw	len	flags
JCXZ sl	E3 r0		2	--------
JECXZ sl [32bit]	67| E3 r0		1+2	--------

Usage
```
     JCXZ    label
     JECXZ   label  (386+ only)
```
Modifies flags

None

Causes execution to branch to "label" if register CX is zero. Uses unsigned comparison.

JE/JZ - Jump Equal / Jump Zero

mnemonics	op xx xx xx xx xx	sw	len	flags
JE ll [386]	0F 84 r0 r1		4	--------
JE sl	74 r0		2	--------

Usage
```
     JE      label
     JZ      label
```
Modifies flags

None

Causes execution to branch to "label" if the Zero Flag is set. Uses unsigned comparison.

*) JE/JZ are different mnemonics for the same instruction

JG/JNLE - Jump Greater / Jump Not Less or Equal

mnemonics	op xx xx xx xx xx	sw	len	flags
JG ll [386]	0F 8F r0 r1		4	--------
JG sl	7F r0		4	--------

Usage
```
     JG      label
     JNLE    label
```
Modifies flags

None

Causes execution to branch to "label" if the Zero Flag is clear or
the Sign Flag equals the Overflow Flag. Signed comparison.

*) JG/JNLE are different mnemonics for the same instruction

JGE/JNL - Jump Greater or Equal / Jump Not Less

mnemonics	op xx xx xx xx xx	sw	len	flags
JGE ll [386]	0F 8D r0 r1		4	--------
JGE sl	7D r0		4	--------

Usage
```
     JGE     label
```

```
        JNL     label
```

Modifies flags

None

Causes execution to branch to "label" if the Sign Flag equals the Overflow Flag. Signed comparison.

*) JGE/JNL are different mnemonics for the same instruction

JL/JNGE - Jump Less / Jump Not Greater or Equal

mnemonics	op xx xx xx xx xx	sw	len	flags
JL ll [386]	0F 8C r0 r1		4	--------
JL sl	7C r0		2	--------

Usage

```
        JL      label
        JNGE    label
```

Modifies flags

None

Causes execution to branch to "label" if the Sign Flag is not equal to Overflow Flag. Unsigned comparison.

*) JL/JNGE are different mnemonics for the same instruction

JLE/JNG - Jump Less or Equal / Jump Not Greater

mnemonics	op xx xx xx xx xx	sw	len	flags
JLE ll [386]	0F 8E r0 r1		4	--------
JLE sl	7E r0		2	--------

Usage

```
        JLE     label
        JNG     label
```

Modifies flags

None

Causes execution to branch to "label" if the Zero Flag is set or the Sign Flag is not equal to the Overflow Flag. Signed comparison.

*) JLE/JNG are different mnemonics for the same instruction

JMP - Unconditional Jump

mnemonics	op xx xx xx xx xx	sw	len	flags
JMP SHORT sl	EB r0		2	--------
JMP np	E9 o0 o1		3	--------
JMP rmw	FF /4 d0 d1		2~4	--------
JMP DWORD PTR [rmw]	FF /5 d0 d1		2~4	--------
JMP FAR PTR fp	EA o0 o1 s0 s1		5	--------

Usage

```
        JMP     target
```

Modifies flags

None

Unconditionally transfers control to "label". Jumps by default are within -32768 to 32767 bytes from the instruction following the jump. NEAR and SHORT jumps cause the IP to be updated while FAR jumps cause CS and IP to be updated.

JNC - Jump Not Carry

mnemonics	op xx xx xx xx xx	sw	len	flags
JNC ll [386]	0F 83 r0 r1		4	--------
JNC sl	73 r0		2	--------

Usage

```
JNC     label
```

Modifies flags

None

Causes execution to branch to "label" if the Carry Flag is clear. Functionally similar to JAE or JNB. Unsigned comparison.

JNE/JNZ - Jump Not Equal / Jump Not Zero

mnemonics	op xx xx xx xx xx	sw	len	flags
JNE ll [386]	0F 85 r0 r1		4	--------
JNE sl	75 r0		2	--------

Usage

```
JNE     label
JNZ     label
```

Modifies flags

None

Causes execution to branch to "label" if the Zero Flag is clear. Unsigned comparison.

*) JNE/JNZ are different mnemonics for the same instruction

JNO - Jump Not Overflow

mnemonics	op xx xx xx xx xx	sw	len	flags
JNO ll [386]	0F 81 r0 r1		4	--------
JNO sl	71 r0		2	--------

Usage

```
JNO     label
```

Modifies flags

None

Causes execution to branch to "label" if the Overflow Flag is clear. Signed comparison.

JNS - Jump Not Signed

mnemonics	op xx xx xx xx xx	sw	len	flags
JNS ll [386]	0F 89 r0 r1		4	--------
JNS sl	79 r0		2	--------

Usage

```
JNS     label
```

Modifies flags

None

Causes execution to branch to "label" if the Sign Flag is clear. Signed comparison.

JNP/JPO - Jump Not Parity / Jump Parity Odd

mnemonics	op xx xx xx xx xx	sw	len	flags
JNP ll [386]	0F 8B r0 r1		4	--------
JNP sl	7B r0		2	--------

Usage

```
JNP     label
JPO     label
```

Modifies flags

None

Causes execution to branch to "label" if the Parity Flag is clear. Unsigned comparison.

*) JNE/JPO are different mnemonics for the same instruction

JO - Jump on Overflow

mnemonics	op xx xx xx xx xx	sw	len	flags
JO ll [386]	0F 80 r0 r1		4	--------
JO sl	70 r0		2	--------

Usage

```
JO  label
```

Modifies flags

None

Causes execution to branch to "label" if the Overflow Flag is set. Signed comparison.

JP/JPE - Jump on Parity / Jump on Parity Even

mnemonics	op xx xx xx xx xx	sw	len	flags
JP ll [386]	0F 8A r0 r1		4	--------
JP sl	7A r0		2	--------

Usage

```
JP      label
JPE     label
```

Modifies flags

None

Causes execution to branch to "label" if the Parity Flag is set. Unsigned comparison.

*) JP/JPE are different mnemonics for the same instruction

JS - Jump Signed

mnemonics	op xx xx xx xx xx	sw	len	flags

JS ll [386]	0F 88 r0 r1		4	--------
JS sl	78 r0		2	--------

Usage
```
    JS      label
```
Modifies flags
    None

Causes execution to branch to "label" if the Sign Flag is set. Signed comparison.

LAHF - Load Register AH From Flags

mnemonics	op xx xx xx xx xx	sw	len	flags
LAHF	9F		1	--------

Usage
```
    LAHF
```
Modifies flags
    None

Copies bits 0-7 of the flags register into AH. This includes flags AF, CF, PF, SF and ZF other bits are undefined.

LAR - Load Access Rights (286+ protected)

mnemonics	op xx xx xx xx xx	sw	len	flags
LAR rw,rmw [286]	0F 02 mr d0 d1		3~5	-----z--

Usage
```
    LAR     dest,src
```
Modifies flags
    ZF

The high byte of the of the destination register is overwritten by the value of the access rights byte and the low order byte is zeroed depending on the selection in the source operand. The Zero Flag is set if the load operation is successful.

LDS - Load Pointer Using DS

mnemonics	op xx xx xx xx xx	sw	len	flags
LDS rw,md	C5 mr d0 d1		2~4	--------

Usage
```
    LDS     dest,src
```
Modifies flags
    None

Loads 32-bit pointer from memory source to destination register and DS. The offset is placed in the destination register and the segment is placed in DS. To use this instruction the word at the lower memory address must contain the offset and the word at the higher address must contain the segment. This simplifies the loading of far pointers from the stack and the interrupt vector table.

LEA - Load Effective Address

mnemonics	op xx xx xx xx xx	sw	len	flags
LEA rw,mw	8D mr d0 d1		2~4	--------

Usage

```
LEA     dest,src
```

Modifies flags

None

Transfers offset address of "src" to the destination register.

LEAVE - Restore Stack for Procedure Exit (80188+)

mnemonics	op xx xx xx xx xx	sw	len	flags
LEAVE [186]	C9		1	--------

Usage

```
LEAVE
```

Modifies flags

None

Releases the local variables created by the previous ENTER instruction by restoring SP and BP to their condition before the procedure stack frame was initialized.

LES - Load Pointer Using ES

mnemonics	op xx xx xx xx xx	sw	len	flags
LES rw,md	C4 mr d0 d1		2~4	--------

Usage

```
LES     dest,src
```

Modifies flags

None

Loads 32-bit pointer from memory source to destination register and ES. The offset is placed in the destination register and the segment is placed in ES. To use this instruction the word at the lower memory address must contain the offset and the word at the higher address must contain the segment. This simplifies the loading of far pointers from the stack and the interrupt vector table.

LFS - Load Pointer Using FS (386+ only)

mnemonics	op xx xx xx xx xx	sw	len	flags
LFS rw,md [386]	0F B4 mr d0 d1		3~5	--------

Usage

```
LFS     dest,src
```

Modifies flags

None

Loads 32-bit pointer from memory source to destination register and FS. The offset is placed in the destination register and the segment is placed in FS. To use this instruction the word at the lower memory address must contain the offset and the word at the higher address must contain the segment. This simplifies the loading of far pointers from the stack and the interrupt vector table.

LGDT - Load Global Descriptor Table (286+ privileged)

mnemonics			op xx xx xx xx xx	sw	len	flags
LGDT	mw	[286]	0F 01 /2 d0 d1		3~5	--------

Usage

```
LGDT    src
```

Modifies flags

None

Loads a value from an operand into the Global Descriptor Table (GDT) register.

LIDT - Load Interrupt Descriptor Table (286+ privileged)

mnemonics			op xx xx xx xx xx	sw	len	flags
LIDT	mw	[286]	0F 01 /3 d0 d1		3~5	--------

Usage

```
LIDT    src
```

Modifies flags

None

Loads a value from an operand into the Interrupt Descriptor Table (IDT) register.

LGS - Load Pointer Using GS (386+ only)

mnemonics			op xx xx xx xx xx	sw	len	flags
LGS	rw,md	[386]	0F B5 mr d0 d1		3~5	--------

Usage

```
LGS     dest,src
```

Modifies flags

None

Loads 32-bit pointer from memory source to destination register and GS. The offset is placed in the destination register and the segment is placed in GS. To use this instruction the word at the lower memory address must contain the offset and the word at the higher address must contain the segment. This simplifies the loading of far pointers from the stack and the interrupt vector table.

LLDT - Load Local Descriptor Table (286+ privileged)

mnemonics			op xx xx xx xx xx	sw	len	flags
LLDT	rmw	[286]	0F 00 /2 d0 d1		3~5	--------

Usage

```
LLDT    src
```

Modifies flags

None

Loads a value from an operand into the Local Descriptor Table Register (LDTR).

LMSW - Load Machine Status Word (286+ privileged)

mnemonics	op xx xx xx xx xx	sw	len	flags
LMSW rmw [286]	0F 01 /6 d0 d1		3~5	--------

Usage
```
LMSW    src
```
Modifies flags

None

Loads the Machine Status Word (MSW) from data found at "src"

LOCK - Lock Bus

mnemonics	op xx xx xx xx xx	sw	len	flags
LOCK	F0		1	--------

Usage
```
LOCK
LOCK: (386+ prefix)
```
Modifies flags

None

This instruction is a prefix that causes the CPU assert bus lock signal during the execution of the next instruction. Used to avoid two processors from updating the same data location. The 286 always asserts lock during an XCHG with memory operands. This should only be used to lock the bus prior to XCHG, MOV, IN and OUT instructions.

LODS - Load String (Byte, Word or Double)

mnemonics	op xx xx xx xx xx	sw	len	flags
LODSB	AC	B	1	--------
LODSW	AD	W	1	--------
LODSD [32bit]	66│ AD	D	1+1	--------

Usage
```
LODS    src
LODSB
LODSW
LODSD  (386+ only)
```
Modifies flags

None

Transfers string element addressed by DS:SI (even if an operand is supplied) to the accumulator. SI is incremented based on the size of the operand or based on the instruction used. If the Direction Flag is set SI is decremented, if the Direction Flag is clear SI is incremented. Use with REP prefixes.

LOOP - Decrement CX and Loop if CX Not Zero

mnemonics	op xx xx xx xx xx	sw	len	flags
LOOP sl	E2 r0		2	--------

Usage
```
LOOP    label
```

Modifies flags
    None

Decrements CX by 1 and transfers control to "label" if CX is not Zero. The "label" operand must be within -128 or 127 bytes of the instruction following the loop instruction

LOOPE/LOOPZ - Loop While Equal / Loop While Zero

mnemonics	op xx xx xx xx xx	sw	len	flags
LOOPE sl	E1 r0		2	--------

Usage

```
    LOOPE   label
    LOOPZ   label
```

Modifies flags
    None

Decrements CX by 1 (without modifying the flags) and transfers control to "label" if CX != 0 and the Zero Flag is set. The "label" operand must be within -128 or 127 bytes of the instruction following the loop instruction.

*) LOOPE/LOOPZ are different mnemonics for the same instruction

LOOPNZ/LOOPNE - Loop While Not Zero / Loop While Not Equal

mnemonics	op xx xx xx xx xx	sw	len	flags
LOOPNZ sl	E0 r0		2	--------

Usage

```
    LOOPNZ  label
    LOOPNE  label
```

Modifies flags
    None

Decrements CX by 1 (without modifying the flags) and transfers control to "label" if CX != 0 and the Zero Flag is clear. The "label" operand must be within -128 or 127 bytes of the instruction following the loop instruction.

*) LOOPNZ/LOOPNE are different mnemonics for the same instruction

LSL - Load Segment Limit (286+ protected)

mnemonics	op xx xx xx xx xx	sw	len	flags
LSL rw,rmw [286]	0F 03 mr d0 d1		3~5	--------

Usage

```
    LSL     dest,src
```

Modifies flags
    ZF

Loads the segment limit of a selector into the destination register if the selector is valid and visible at the current privilege level. If loading is successful the Zero Flag is set, otherwise it is cleared.

LSS - Load Pointer Using SS (386+ only)

mnemonics	op xx xx xx xx xx	sw	len	flags
LSS rw,md [386]	0F B2 mr d0 d1		3~5	--------

Usage

```
    LSS     dest,src
```

Modifies flags

None

Loads 32-bit pointer from memory source to destination register and SS. The offset is placed in the destination register and the segment is placed in SS. To use this instruction the word at the lower memory address must contain the offset and the word at the higher address must contain the segment. This simplifies the loading of far pointers from the stack and the interrupt vector table.

LTR - Load Task Register (286+ privileged)

mnemonics	op xx xx xx xx xx	sw	len	flags
LTR rmw [286]	0F 00 /3 d0 d1		3~5	--------

Usage

```
    LTR     src
```

Modifies flags

None

Loads the current task register with the value specified in "src".

MOV - Move Byte or Word

mnemonics	op xx xx xx xx xx	sw	len	flags
MOV AL,rmb	A0 d0 d1	B	3	--------
MOV AX,rmw	A1 d0 d1	W	3	--------
MOV AL,ib	B0 i0	B	2	--------
MOV AH,ib	B4 i0	B	2	--------
MOV AX,iw	B8 i0 i1	W	3	--------
MOV CL,ib	B1 i0	B	2	--------
MOV CH,ib	B5 i0	B	2	--------
MOV CX,iw	B9 i0 i1	W	3	--------
MOV DL,ib	B2 i0	B	2	--------
MOV DH,ib	B6 i0	B	2	--------
MOV DX,iw	BA i0 i1	W	3	--------
MOV BL,ib	B3 i0	B	2	--------
MOV BH,ib	B7 i0	B	2	--------
MOV BX,iw	BB i0 i1	W	3	--------
MOV SP,iw	BC i0 i1	W	3	--------
MOV BP,iw	BD i0 i1	W	3	--------
MOV SI,iw	BE i0 i1	W	3	--------
MOV DI,iw	BF i0 i1	W	3	--------
MOV cr,rd [386]	0F 22 mr		3	--------
MOV rd,cr [386]	0F 20 mr		3	--------

MOV dr,rd [386]	0F 23 mr		3	--------
MOV rd,dr [386]	0F 21 mr		3	--------
MOV tr,rd [386]	0F 26 mr		2	--------
MOV rd,tr [386]	0F 24 mr		3	--------
MOV rb,rmb	8A mr d0 d1	B	2~4	--------
MOV rmb,rb	88 mr d0 d1	B	2~4	--------
MOV rmb,AL	A2 d0 d1	B	3	--------
MOV rmw,AX	A3 d0 d1	W	3	--------
MOV rmb,ib	C6 mr d0 d1 i0	B	3~5	--------
MOV rmw,iw	C7 mr d0 d1 i0 i1	W	4~6	--------
MOV rmw,rw	89 mr d0 d1	W	2~4	--------
MOV rw,rmw	8B mr d0 d1	W	2~4	--------
MOV rmw,sr	8C mr d0 d1		2~4	--------
MOV sr,rmw	8E mr d0 d1		2~4	--------

Usage

```
MOV     dest,src
```

Modifies flags

None

Copies byte or word from the source operand to the destination operand. If the destination is SS interrupts are disabled except on early buggy 808x CPUs. Some CPUs disable interrupts if the destination is any of the segment registers

MOVS - Move String (Byte or Word)

mnemonics	op xx xx xx xx xx	sw	len	flags
MOVSB	A4	B	1	--------
MOVSW	A5	W	1	--------
MOVSD [32bit]	66| A5	D	1+1	--------

Usage

```
MOVS    dest,src
MOVSB
MOVSW
MOVSD  (386+ only)
```

Modifies flags

None

Copies data from addressed by DS:SI (even if operands are given) to the location ES:DI destination and updates SI and DI based on the size of the operand or instruction used. SI and DI are incremented when the Direction Flag is cleared and decremented when the Direction Flag is Set. Use with REP prefixes.

MOVSX - Move with Sign Extend (386+ only)

mnemonics	op xx xx xx xx xx	sw	len	flags
MOVSX rw,rmb [386]	0F BE mr d0 d1	B	3~5	--------
MOVSX rd,rmw [386]	0F BF mr d0 d1	W	3~5	--------

Usage

```
MOVSX   dest,src
```

Modifies flags
None

Copies the value of the source operand to the destination register with the sign extended.

MOVZX - Move with Zero Extend (386+ only)

mnemonics	op xx xx xx xx xx	sw	len	flags
MOVZX rw,rmb [386]	0F B6 mr d0 d1	B	3~5	--------
MOVZX rd,rmw [386]	0F B7 mr d0 d1	B	3~5	--------

Usage
```
MOVZX   dest,src
```
Modifies flags
None

Copies the value of the source operand to the destination register with the zeroes extended.

MUL - Unsigned Multiply

mnemonics	op xx xx xx xx xx	sw	len	flags
MUL rmb	F6 /4 d0 d1	B	2~4	o---szap
MUL rmw	F7 /4 d0 d1	W	2~4	o---szap

Usage
```
MUL     src
```
Modifies flags
CF OF (AF,PF,SF,ZF undefined)

Unsigned multiply of the accumulator by the source. If "src" is a byte value, then AL is used as the other multiplicand and the result is placed in AX. If "src" is a word value, then AX is multiplied by "src" and DX:AX receives the result. If "src" is a double word value, then EAX is multiplied by "src" and EDX:EAX receives the result. The 386+ uses an early out algorithm which makes multiplying any size value in EAX as fast as in the 8 or 16 bit registers.

NEG - Two's Complement Negation

mnemonics	op xx xx xx xx xx	sw	len	flags
NEG rmb	F6 /3 d0 d1	B	2~4	o---szap
NEG rmw	F7 /3 d0 d1	W	2~4	o---szap

Usage
```
NEG     dest
```
Modifies flags
AF CF OF PF SF ZF

Subtracts the destination from 0 and saves the 2s complement of "dest" back into "dest".

NOP - No Operation

mnemonics	op xx xx xx xx xx	sw	len	flags
NOP	90		1	--------

Usage
NOP
Modifies flags
None

This is a do nothing instruction. It results in occupation of both space and time and is most useful for patching :-) code [segments].

NOT - One's Compliment Negation (Logical NOT)

mnemonics	op xx xx xx xx xx	sw	len	flags
NOT rmb	F6 /2 d0 d1	B	2~4	--------
NOT rmw	F7 /2 d0 d1	W	2~4	--------

Usage
NOT dest
Modifies flags
None

Inverts the bits of the "dest" operand forming the 1s complement.

OR - Inclusive Logical OR

mnemonics	op xx xx xx xx xx	sw	len	flags
OR AL,ib	0C i0	B	2	o---szap
OR AX,iw	0D i0 i1	W	3	o---szap
OR rb,rmb	0A mr d0 d1	B	2~4	o---szap
OR rw,rmw	0B mr d0 d1	W	2~4	o---szap
OR rmb,ib	80 /1 d0 d1 i0	NB	3~5	o---szap
OR rmw,iw	81 /1 d0 d1 i0 i1	NW	4~6	o---szap
OR rmw,ib	83 /1 d0 d1 i0	EW	3~5	o---szap
OR rmb,rb	08 mr d0 d1	B	2~4	o---szap
OR rmw,rw	09 mr d0 d1	W	2~4	o---szap

Usage
OR dest,src
Modifies flags
CF OF PF SF ZF (AF undefined)

Logical inclusive OR of the two operands returning the result in the destination. Any bit set in either operand will be set in the destination.

OUT - Output Data to Port

mnemonics	op xx xx xx xx xx	sw	len	flags
OUT DX,AL	EE	B	1	--------
OUT DX,AX	EF	W	1	--------
OUT ib,AL	E6 i0	B	2	--------
OUT ib,AX	E7 i0	W	2	--------

Usage

```
        OUT     port,accum
```

Modifies flags

        None

Transfers byte in AL,word in AX or dword in EAX to the specified hardware port address. If the port number is in the range of 0-255 it can be specified as an immediate. If greater than 255 then the port number must be specified in DX. Since the PC only decodes 10 bits of the port address, values over 1023 can only be decoded by third party vendor equipment and also map to the port range 0-1023.

OUTS - Output String to Port (80188+ only)

mnemonics	op xx xx xx xx xx	sw	len	flags
OUTSB [186]	6E	B	1	--------
OUTSW [186]	6F	W	1	--------
OUTSD [32bit]	66| 6F	D	1+1	--------

Usage

```
        OUTS    port,src
        OUTSB
        OUTSW
        OUTSD  (386+ only)
```

Modifies flags

        None

Transfers a byte, word or doubleword from "src" to the hardware port specified in DX. For instructions with no operands the "src" is located at DS:SI and SI is incremented or decremented by the size of the operand or the size dictated by the instruction format. When the Direction Flag is set SI is decremented, when clear, SI is incremented. If the port number is in the range of 0-255 it can be specified as an immediate. If greater than 255 then the port number must be specified in DX. Since the PC only decodes 10 bits of the port address, values over 1023 can only be decoded by third party vendor equipment and also map to the port range 0-1023.

POP - Pop Word off Stack

mnemonics	op xx xx xx xx xx	sw	len	flags
POP AX	58		1	--------
POP CX	59		1	--------
POP DX	5A		1	--------
POP BX	5B		1	--------
POP SP	5C		1	--------
POP BP	5D		1	--------
POP SI	5E		1	--------
POP DI	5F		1	--------
POP ES	07		1	--------
POP SS	17		1	--------
POP DS	1F		1	--------
POP FS [386]	0F A1		2	--------
POP GS [386]	0F A9		2	--------

POP rmw	8F mr d0 d1		2~4	--------

Usage

```
POP     dest
```

Modifies flags

None

Transfers word at the current stack top (SS:SP) to the destination then increments SP by two to point to the new stack top. CS is not a valid destination.

POPA/POPAD - Pop All Registers onto Stack (80188+ only)

mnemonics	op xx xx xx xx xx	sw	len	flags
POPA [186]	61		1	--------
POPAD [32bit]	66│ 61		1+1	--------

Usage

```
POPA
POPAD  (386+ only)
```

Modifies flags

None

Pops the top 8 words off the stack into the 8 general purpose 16/32 bit registers. Registers are popped in the following order: (E)DI, (E)SI, (E)BP, (E)SP, (E)DX, (E)CX and (E)AX. The (E)SP value popped from the stack is actually discarded.

POPF/POPFD - Pop Flags off Stack

mnemonics	op xx xx xx xx xx	sw	len	flags
POPF	9D		1	oditszap
POPFD [32bit]	66│ 9D		1+1	oditszap

Usage

```
POPF
POPFD  (386+ only)
```

Modifies flags

all flags

Pops word/doubleword from stack into the Flags Register and then increments SP by 2 (for POPF) or 4 (for POPFD).

PUSH - Push Word onto Stack

mnemonics	op xx xx xx xx xx	sw	len	flags
PUSH AX	50		1	--------
PUSH CX	51		1	--------
PUSH DX	52		1	--------
PUSH BX	53		1	--------
PUSH SP	54		1	--------
PUSH BP	55		1	--------
PUSH SI	56		1	--------
PUSH DI	57		1	--------
PUSH ES	06		1	--------

PUSH CS	0E		1	--------
PUSH SS	16		1	--------
PUSH DS	1E		1	--------
PUSH FS [386]	0F A0		2	--------
PUSH GS [386]	0F A8		2	--------
PUSH ib [186]	6A i0	E	2	--------
PUSH iw [186]	68 i0 i1	N	3	--------
PUSH rmw	FF /6 d0 d1		2~4	--------

Usage

```
PUSH    src
PUSH    immed   (80188+ only)
```

Modifies flags

None

Decrements SP by the size of the operand (two or four, byte values are sign extended) and transfers one word from source to the stack top (SS:SP).

PUSHA/PUSHAD - Push All Registers onto Stack (80188+ only)

mnemonics	op xx xx xx xx xx	sw	len	flags
PUSHA [186]	60		1	--------
PUSHAD [32bit]	66| 60		1+1	--------

Usage

```
PUSHA
PUSHAD  (386+ only)
```

Modifies flags

None

Pushes all general purpose registers onto the stack in the following order: (E)AX, (E)CX, (E)DX, (E)BX, (E)SP, (E)BP, (E)SI, (E)DI. The value of SP is the value before the actual push of SP.

PUSHF/PUSHFD - Push Flags onto Stack

mnemonics	op xx xx xx xx xx	sw	len	flags
PUSHF	9C		1	--------
PUSHFD [32bit]	66| 9C		1+1	--------

Usage

```
PUSHF
PUSHFD  (386+ only)
```

Modifies flags

None

Transfers the Flags Register onto the stack. PUSHF saves a 16 bit value while PUSHFD saves a 32 bit value.

RCL - Rotate Through Carry Left

mnemonics	op xx xx xx xx xx	sw	len	flags
RCL rmb,1	D0 /2 d0 d1	B	2~4	o-------

RCL rmb,CL	D2 /2 d0 d1	B	2~4	o-------
RCL rmb,ib [186]	C0 /2 d0 d1 i0	B	3~5	o-------
RCL rmw,1	D1 /2 d0 d1	W	2~4	o-------
RCL rmw,CL	D3 /2 d0 d1	W	2~4	o-------
RCL rmw,ib [186]	C1 /2 d0 d1 i0	W	3~5	o-------

Usage

```
        RCL     dest,count
```

Modifies flags

CF OF

Rotates the bits in the destination to the left "count" times with all data pushed out the left side re-entering on the right. The Carry Flag holds the last bit rotated out.

RCR - Rotate Through Carry Right

mnemonics	op xx xx xx xx xx	sw	len	flags
RCR rmb,1	D0 /3 d0 d1	B	2~4	o-------
RCR rmb,CL	D2 /3 d0 d1	B	2~4	o-------
RCR rmb,ib [186]	C0 /3 d0 d1 i0	B	3~5	o-------
RCR rmw,1	D1 /3 d0 d1	W	2~4	o-------
RCR rmw,CL	D3 /3 d0 d1	W	2~4	o-------
RCR rmw,ib [186]	C1 /3 d0 d1 i0	W	3~5	o-------

Usage

```
        RCR     dest,count
```

Modifies flags

CF OF

Rotates the bits in the destination to the right "count" times with all data pushed out the right side re-entering on the left. The Carry Flag holds the last bit rotated out.

REP - Repeat String Operation

mnemonics	op xx xx xx xx xx	sw	len	flags
REP	F3		1	-----z--

Usage

```
        REP
```

Modifies flags

None

Repeats execution of string instructions while CX != 0. After each string operation, CX is decremented and the Zero Flag is tested. The combination of a repeat prefix and a segment override on CPU's before the 386 may result in errors if an interrupt occurs before CX=0. The following code shows code that is susceptible to this and how to avoid it:

```
again: rep movs byte ptr ES:[DI],ES:[SI] ; vulnerable instr.
jcxz next ; continue if REP successful
loop again ; interrupt goofed count
next:
```

REPE/REPZ - Repeat Equal / Repeat Zero

mnemonics	op xx xx xx xx xx	sw	len	flags
REPE	F3		1	--------

Usage

```
        REPE
        REPZ
```

Modifies flags

None

Repeats execution of string instructions while CX != 0 and the Zero Flag is set. CX is decremented and the Zero Flag tested after each string operation. The combination of a repeat prefix and a segment override on processors other than the 386 may result in errors if an interrupt occurs before CX=0.

*) REPE/REPZ are different mnemonics for the same instruction

REPNE/REPNZ - Repeat Not Equal / Repeat Not Zero

mnemonics	op xx xx xx xx xx	sw	len	flags
REPNE	F2		1	-----z--

Usage

```
        REPNE
        REPNZ
```

Modifies flags

None

Repeats execution of string instructions while CX != 0 and the Zero Flag is clear. CX is decremented and the Zero Flag tested after each string operation. The combination of a repeat prefix and a segment override on processors other than the 386 may result in errors if an interrupt occurs before CX=0.

*) REPNE/REPNZ are different mnemonics for the same instruction

RET/RETF - Return From Procedure

mnemonics	op xx xx xx xx xx	sw	len	flags
RET	C3		1	--------
RET iw	C2 i0 i1		3	--------
RETF	CB		1	--------
RETF iw	CA i0 i1		3	--------

Usage

```
        RET     nBytes
        RETF    nBytes
        RETN    nBytes
```

Modifies flags

None

Transfers control from a procedure back to the instruction address saved on the stack. "n bytes" is an optional number of bytes to release. Far returns pop the IP followed by the CS, while near returns pop only the IP register.

ROL - Rotate Left

mnemonics	op xx xx xx xx xx	sw	len	flags
ROL rmb,1	D0 /0 d0 d1	B	2~4	o-------
ROL rmb,CL	D2 /0 d0 d1	B	2~4	o-------
ROL rmb,ib [186]	C0 /0 d0 d1 i0	B	3~5	o-------
ROL rmw,1	D1 /0 d0 d1	W	2~4	o-------
ROL rmw,CL	D3 /0 d0 d1	W	2~4	o-------
ROL rmw,ib [186]	C1 /0 d0 d1 i0	W	3~5	o-------

Usage
```
        ROL     dest,count
```
Modifies flags
        CF OF

Rotates the bits in the destination to the left "count" times with all data pushed out the left side re-entering on the right. The Carry Flag will contain the value of the last bit rotated out.

ROR - Rotate Right

mnemonics	op xx xx xx xx xx	sw	len	flags
ROR rmb,1	D0 /1 d0 d1	B	2~4	o-------
ROR rmb,CL	D2 /1 d0 d1	B	2~4	o-------
ROR rmb,ib [186]	C0 /1 d0 d1 i0	B	3~5	o-------
ROR rmw,1	D1 /1 d0 d1	W	2~4	o-------
ROR rmw,CL	D3 /1 d0 d1	W	2~4	o-------
ROR rmw,ib [186]	C1 /1 d0 d1 i0	W	3~5	o-------

Usage
```
        ROR     dest,count
```
Modifies flags
        CF OF

Rotates the bits in the destination to the right "count" times with all data pushed out the right side re-entering on the left. The Carry Flag will contain the value of the last bit rotated out.

SAHF - Store AH Register into FLAGS

mnemonics	op xx xx xx xx xx	sw	len	flags
SAHF	9E		1	----szap

Usage
```
        SAHF
```
Modifies flags
        AF CF PF SF ZF

Transfers bits 0-7 of AH into the Flags Register. This includes AF, CF, PF, SF and ZF.

SAL/SHL - Shift Arithmetic Left / Shift Logical Left

mnemonics	op xx xx xx xx xx	sw	len	flags
SAL rmb,1	D0 /4 d0 d1	B	2~4	o-------

SAL rmb,CL	D2 /4 d0 d1	B	2~4	o-------
SAL rmb,ib [186]	C0 /4 d0 d1 i0	B	3~5	o-------
SAL rmw,1	D1 /4 d0 d1	W	2~4	o-------
SAL rmw,CL	D3 /4 d0 d1	W	2~4	o-------
SAL rmw,ib [186]	C1 /4 d0 d1 i0	W	3~5	o-------
SHL rmb,1	D0 /4 d0 d1	B	2~4	o-------
SHL rmb,CL	D2 /4 d0 d1	B	2~4	o-------
SHL rmb,ib [186]	C0 /4 d0 d1 i0	B	3~5	o-------
SHL rmw,1	D1 /4 d0 d1	W	2~4	o-------
SHL rmw,CL	D3 /4 d0 d1	W	2~4	o-------
SHL rmw,ib [186]	C1 /4 d0 d1 i0	W	3~5	o-------

Usage

```
SAL     dest,count
SHL     dest,count
```

Modifies flags

CF OF PF SF ZF (AF undefined)

Shifts the destination left by "count" bits with zeroes shifted in on right. The Carry Flag contains the last bit shifted out.

SAR - Shift Arithmetic Right

mnemonics	op xx xx xx xx xx	sw	len	flags
SAR rmb,CL	D2 /7 d0 d1	B	2~4	o-------
SAR rmb,ib [186]	C0 /7 d0 d1 i0	B	3~5	o-------
SAR rmw,1	D1 /7 d0 d1	W	2~4	o-------
SAR rmw,CL	D3 /7 d0 d1	W	2~4	o-------
SAR rmw,ib [186]	C1 /7 d0 d1 i0	W	3~5	o-------

Usage

```
SAR     dest,count
```

Modifies flags

CF OF PF SF ZF (AF undefined)

Shifts the destination right by "count" bits with the current sign bit replicated in the leftmost bit. The Carry Flag contains the last bit shifted out.

SBB - Subtract with Borrow

mnemonics	op xx xx xx xx xx	sw	len	flags
SBB AL,ib	1C i0	B	2	o---szap
SBB AX,iw	1D i0 i1	W	3	o---szap
SBB rb,rmb	1A mr d0 d1	B	2~4	o---szap
SBB rw,rmw	1B mr d0 d1	W	2~4	o---szap
SBB rmb,ib	80 /3 d0 d1 i0	NB	3~5	o---szap
SBB rmw,iw	81 /3 d0 d1 i0 i1	NW	4~6	o---szap
SBB rmw,ib	83 /3 d0 d1 i0	EW	3~5	o---szap
SBB rmb,rb	18 mr d0 d1	B	2~4	o---szap
SBB rmw,rw	19 mr d0 d1	W	2~4	o---szap

Usage

```
        SBB     dest,src
```

Modifies flags

        AF CF OF PF SF ZF

Subtracts the source from the destination, and subtracts 1 extra if the Carry Flag is set. Results are returned in "dest".

SCAS - Scan String (Byte, Word or Doubleword)

mnemonics	op xx xx xx xx xx	sw	len	flags
SCASB	AE	B	1	o---szap
SCASW	AF	W	1	o---szap
SCASD [32bit]	66| AF	D	1+1	o---szap

Usage

```
        SCAS    string
        SCASB
        SCASW
        SCASD   (386+ only)
```

Modifies flags

        AF CF OF PF SF ZF

Compares value at ES:DI (even if operand is specified) from the accumulator and sets the flags similar to a subtraction. DI is incremented/decremented based on the instruction format (or operand size) and the state of the Direction Flag. Use with REP prefixes.

SETAE/SETNB - Set if Above or Equal / Set if Not Below (unsigned, 386+ only)

mnemonics	op xx xx xx xx xx	sw	len	flags
SETAE rmb [386]	0F 93 mr d0 d1		3~5	--------

Usage

```
        SETAE   dest
        SETNB   dest
```

Modifies flags

        none

Sets the byte in the operand to 1 if the Carry Flag is clear otherwise sets the operand to 0.

*) SETAE/SETNB are different mnemonics for the same instruction

SETB/SETNAE - Set if Below / Set if Not Above or Equal (unsigned, 386+ only)

mnemonics	op xx xx xx xx xx	sw	len	flags
SETB rmb [386]	0F 92 mr d0 d1		3~5	--------

Usage

```
        SETB    dest
        SETNAE  dest
```

Modifies flags

        none

Sets the byte in the operand to 1 if the Carry Flag is set otherwise sets the operand to 0.

*) SETB/SETAE are different mnemonics for the same instruction

SETBE/SETNA - Set if Below or Equal / Set if Not Above (unsigned, 386+ only)

mnemonics	op xx xx xx xx xx	sw	len	flags
SETBE rmb [386]	0F 96 mr d0 d1		3~5	--------

Usage
```
SETBE   dest
SETNA   dest
```
Modifies flags
none

Sets the byte in the operand to 1 if the Carry Flag or the Zero Flag is set, otherwise sets the operand to 0.

*) SETBE/SETNA are different mnemonics for the same instruction

SETE/SETZ - Set if Equal / Set if Zero (386+ only)

mnemonics	op xx xx xx xx xx	sw	len	flags
SETZ rmb [386]	0F 94 mr d0 d1		3~5	--------

Usage
```
SETE    dest
SETZ    dest
```
Modifies flags
none

Sets the byte in the operand to 1 if the Zero Flag is set, otherwise sets the operand to 0.

*) SETE/SETZ are different mnemonics for the same instruction

SETNE/SETNZ - Set if Not Equal / Set if Not Zero (386+ only)

mnemonics	op xx xx xx xx xx	sw	len	flags
SETNE rmb [386]	0F 95 mr d0 d1		3~5	--------

Usage
```
SETNE   dest
SETNZ   dest
```
Modifies flags
none

Sets the byte in the operand to 1 if the Zero Flag is clear, otherwise sets the operand to 0.

*) SETNE/SETNZ are different mnemonics for the same instruction

SETL/SETNGE - Set if Less / Set if Not Greater or Equal (signed, 386+ only)

mnemonics	op xx xx xx xx xx	sw	len	flags
SETL rmb [386]	0F 9C mr d0 d1		3~5	--------

Usage

```
      SETL    dest
      SETNGE  dest
```

Modifies flags

```
      none
```

Sets the byte in the operand to 1 if the Sign Flag is not equal to the Overflow Flag, otherwise sets the operand to 0.

*) SETL/SETNGE are different mnemonics for the same instruction

SETGE/SETNL - Set if Greater or Equal / Set if Not Less (signed, 386+ only)

mnemonics	op xx xx xx xx xx	sw	len	flags
SETGE rmb [386]	0F 9D mr d0 d1		3~5	--------

Usage

```
      SETGE   dest
      SETNL   dest
```

Modifies flags

```
      none
```

Sets the byte in the operand to 1 if the Sign Flag equals the Overflow Flag, otherwise sets the operand to 0.

*) SETGE/SETNL are different mnemonics for the same instruction

SETLE/SETNG - Set if Less or Equal / Set if Not greater or Equal (signed, 386+ only)

mnemonics	op xx xx xx xx xx	sw	len	flags
SETLE rmb [386]	0F 9E mr d0 d1		3~5	--------

Usage

```
      SETLE   dest
      SETNG   dest
```

Modifies flags

```
      none
```

Sets the byte in the operand to 1 if the Zero Flag is set or the Sign Flag is not equal to the Overflow Flag, otherwise sets the operand to 0.

*) SETLE/SETNG are different mnemonics for the same instruction

SETG/SETNLE - Set if Greater / Set if Not Less or Equal (signed, 386+ only)

mnemonics	op xx xx xx xx xx	sw	len	flags
SETG rmb [386]	0F 9F mr d0 d1		3~5	--------

Usage

```
      SETG    dest
      SETNLE  dest
```

Modifies flags

```
      none
```

Sets the byte in the operand to 1 if the Zero Flag is clear or the

Sign Flag equals to the Overflow Flag, otherwise sets the operand to 0.

*) SETG/SETNLE are different mnemonics for the same instruction

SETS - Set if Signed (386+ only)

mnemonics	op xx xx xx xx xx	sw	len	flags
SETS rmb [386]	0F 98 mr d0 d1		3~5	--------

Usage
```
    SETS    dest
```
Modifies flags
none

Sets the byte in the operand to 1 if the Sign Flag is set, otherwise sets the operand to 0.

SETNS - Set if Not Signed (386+ only)

mnemonics	op xx xx xx xx xx	sw	len	flags
SETNS rmb [386]	0F 99 mr d0 d1		3~5	--------

Usage
```
    SETNS   dest
```
Modifies flags
none

Sets the byte in the operand to 1 if the Sign Flag is clear, otherwise sets the operand to 0.

SETC - Set if Carry (386+ only)

mnemonics	op xx xx xx xx xx	sw	len	flags
SETC rmb [386]	0F 92 mr d0 d1		3~5	--------

Usage
```
    SETC    dest
```
Modifies flags
none

Sets the byte in the operand to 1 if the Carry Flag is set, otherwise sets the operand to 0.

SETNC - Set if Not Carry (386+ only)

mnemonics	op xx xx xx xx xx	sw	len	flags
SETNC rmb [386]	0F 93 mr d0 d1		3~5	--------

Usage
```
    SETNC   dest
```
Modifies flags
none

Sets the byte in the operand to 1 if the Carry Flag is clear, otherwise sets the operand to 0.

SETO - Set if Overflow (386+ only)

mnemonics	op xx xx xx xx xx	sw	len	flags
SETO rmb [386]	0F 90 mr d0 d1		3~5	--------

Usage
```
SETO    dest
```
Modifies flags
none

Sets the byte in the operand to 1 if the Overflow Flag is set,
otherwise sets the operand to 0.

SETNO - Set if Not Overflow (386+ only)

mnemonics	op xx xx xx xx xx	sw	len	flags
SETNO rmb [386]	0F 91 mr d0 d1		3~5	--------

Usage
```
SETNO   dest
```
Modifies flags
none

Sets the byte in the operand to 1 if the Overflow Flag is clear,
otherwise sets the operand to 0.

SETP/SETPE - Set if Parity / Set if Parity Even (386+ only)

mnemonics	op xx xx xx xx xx	sw	len	flags
SETP rmb [386]	0F 9A mr d0 d1		3~5	--------

Usage
```
SETP    dest
SETPE   dest
```
Modifies flags
none

Sets the byte in the operand to 1 if the Parity Flag is set,
otherwise sets the operand to 0.

*) SETP/SETE are different mnemonics for the same instruction

SETNP/SETPO - Set if No Parity / Set if Parity Odd (386+ only)

mnemonics	op xx xx xx xx xx	sw	len	flags
SETNP rmb [386]	0F 9B mr d0 d1		3~5	--------

Usage
```
SETNP   dest
SETPO   dest
```
Modifies flags
none

Sets the byte in the operand to 1 if the Parity Flag is clear,
otherwise sets the operand to 0.

*) SETNP/SETPO are different mnemonics for the same instruction

SGDT - Store Global Descriptor Table (286+ privileged)

mnemonics	op xx xx xx xx xx	sw	len	flags
SGDT m6 [286]	0F 01 /0 d0 d1		3~5	--------

Usage

```
SGDT    dest
```

Modifies flags

none

Stores the Global Descriptor Table (GDT) Register into the specified operand.

SIDT - Store Interrupt Descriptor Table (286+ privileged)

mnemonics	op xx xx xx xx xx	sw	len	flags
SIDT m6 [286]	0F 01 /1 d0 d1		3~5	--------

Usage

```
SIDT    dest
```

Modifies flags

none

Stores the Interrupt Descriptor Table (IDT) Register into the specified operand.

SHR - Shift Logical Right

mnemonics	op xx xx xx xx xx	sw	len	flags
SHR rmb,1	D0 /5 d0 d1	B	2~4	o-------
SHR rmb,CL	D2 /5 d0 d1	B	2~4	o-------
SHR rmb,ib [186]	C0 /5 d0 d1 i0	B	3~5	o-------
SHR rmw,1	D1 /5 d0 d1	W	2~4	o-------
SHR rmw,CL	D3 /5 d0 d1	W	2~4	o-------
SHR rmw,ib [186]	C1 /5 d0 d1 i0	W	3~5	o-------

Usage

```
SHR     dest,count
```

Modifies flags

CF OF PF SF ZF (AF undefined)

Shifts the destination right by "count" bits with zeroes shifted in on the left. The Carry Flag contains the last bit shifted out.

SHLD/SHRD - Double Precision Shift (386+ only)

mnemonics	op xx xx xx xx xx	sw	len	flags
SHLD rmw,rw,CL [386]	0F A5 mr d0 d1		3~5	o---szap
SHLD rmw,rw,ib [386]	0F A4 mr d0 d1 i0		4~6	o---szap
SHRD rmw,rw,CL [386]	0F AD mr d0 d1		3~5	o---szap
SHRD rmw,rw,ib [386]	0F AC mr d0 d1 i0		4~6	o---szap

Usage

```
    SHLD    dest,src,count
    SHRD    dest,src,count
```

Modifies flags

    CF PF SF ZF (OF,AF undefined)

SHLD shifts "dest" to the left "count" times and the bit positions opened are filled with the most significant bits of "src". SHRD shifts "dest" to the right "count" times and the bit positions opened are filled with the least significant bits of the second operand. Only the 5 lower bits of "count" are used.

SLDT - Store Local Descriptor Table (286+ privileged)

mnemonics	op xx xx xx xx xx	sw	len	flags
SLDT mw [286]	0F 00 /0 d0 d1		3~5	--------

Usage

```
    SLDT    dest
```

Modifies flags

    none

Stores the Local Descriptor Table (LDT) Register into the specified operand.

SMSW - Store Machine Status Word (286+ privileged)

mnemonics	op xx xx xx xx xx	sw	len	flags
SMSW rmw [286]	0F 01 /4 d0 d1		3~5	--------

Usage

```
    SMSW    dest
```

Modifies flags

    none

Store Machine Status Word (MSW) into "dest".

STC - Set Carry

mnemonics	op xx xx xx xx xx	sw	len	flags
STC	F9		1	--------

Usage

```
    STC
```

Modifies flags

    CF

Sets the Carry Flag to 1.

STD - Set Direction Flag

mnemonics	op xx xx xx xx xx	sw	len	flags
STD	FD		1	-1------

Usage

```
    STD
```

Modifies flags
DF

Sets the Direction Flag to 1 causing string instructions to auto-decrement SI and DI instead of auto-increment.

STI - Set Interrupt Flag (Enable Interrupts)

mnemonics	op xx xx xx xx xx	sw	len	flags
STI	FB		1	--1-----

Usage
```
        STI
```
Modifies flags
IF

Sets the Interrupt Flag to 1, enabling recognition of all CPU hardware interrupts.

STOS - Store String (Byte, Word or Doubleword)

mnemonics	op xx xx xx xx xx	sw	len	flags
STOSB	AA	B	1	--------
STOSW	AB	W	1	--------
STOSD [32bit]	66| AB	D	1+1	--------

Usage
```
        STOS    dest
        STOSB
        STOSW
        STOSD   (386+ only)
```
Modifies flags
None

Stores value in accumulator to location at ES:(E)DI (even if operand is given). (E)DI is incremented/decremented based on the size of the operand (or instruction format) and the state of the Direction Flag. Use with REP prefixes.

STR - Store Task Register (286+ privileged)

mnemonics	op xx xx xx xx xx	sw	len	flags
STR rmw [286]	0F 01 /1 d0 d1		3~5	--------

Usage
```
        STR     dest
```
Modifies flags
None

Stores the current Task Register to the specified operand.

SUB - Subtract

mnemonics	op xx xx xx xx xx	sw	len	flags
SUB AL,ib	2C i0	B	2	o---szap
SUB AX,iw	2D i0 i1	W	3	o---szap

SUB rb,rmb	2A mr d0 d1	B	2~4	o---szap
SUB rw,rmw	2B mr d0 d1	W	2~4	o---szap
SUB rmb,ib	80 /5 d0 d1 i0	NB	3~5	o---szap
SUB rmw,iw	81 /5 d0 d1 i0 i1	NW	4~6	o---szap
SUB rmw,ib	83 /5 d0 d1 i0	EW	3~5	o---szap
SUB rmb,rb	28 mr d0 d1	B	2~4	o---szap
SUB rmw,rw	29 mr d0 d1	W	2~4	o---szap

Usage

```
        SUB     dest,src
```

Modifies flags

AF CF OF PF SF ZF

The source is subtracted from the destination and the result is stored in the destination.

TEST - Test For Bit Pattern

mnemonics	op xx xx xx xx xx	sw	len	flags
TEST AL,ib	A8 i0	B	2	0---szap
TEST AX,iw	A9 i0 i1	W	3	0---szap
TEST rmb,ib	F6 /0 d0 d1 i0	B	3~5	0---szap
TEST rmw,iw	F7 /0 d0 d1 i0 i1	W	4~6	0---szap
TEST rmb,rmb	84 mr d0 d1	B	2~4	0---szap
TEST rmw,rmw	85 mr d0 d1	W	2~4	0---szap

Usage

```
        TEST    dest,src
```

Modifies flags

CF OF PF SF ZF (AF undefined)

Performs a logical AND of the two operands updating the flags register without saving the result.

VERR - Verify Read (286+ protected)

mnemonics	op xx xx xx xx xx	sw	len	flags
VERR rmw [286]	0F 00 /4 d0 d1		3~5	-----z--

Usage

```
        VERR    src
```

Modifies flags

ZF

Verifies the specified segment selector is valid and is readable at the current privilege level. If the segment is readable, the Zero Flag is set, otherwise it is cleared.

VERW - Verify Write (286+ protected)

mnemonics	op xx xx xx xx xx	sw	len	flags
VERW rmw [286]	0F 00 /5 d0 d1		3~5	-----z--

Usage

```
        VERW    src
```

Modifies flags

ZF

Verifies the specified segment selector is valid and is ratable
at the current privilege level. If the segment is writable,
the Zero Flag is set, otherwise it is cleared.

WAIT/FWAIT - Event Wait

mnemonics	op xx xx xx xx xx	sw	len	flags
WAIT	9B		1	--------

Usage

```
        WAIT
        FWAIT
```

Modifies flags

None

CPU enters wait state until the coprocessor signals it has finished
it's operation. This instruction is used to prevent the CPU from
accessing memory that may be temporarily in use by the coprocessor.
WAIT and FWAIT are identical.

WBINVD - Write-Back and Invalidate Cache

mnemonics	op xx xx xx xx xx	sw	len	flags
WBINVD [486]	0F 09		2	--------

Usage

```
        WBINVD
```

Modifies flags

None

Flushes internal cache, then signals the external cache to write
back current data followed by a signal to flush the external cache.

XCHG - Exchange

mnemonics	op xx xx xx xx xx	sw	len	flags
XCHG AX,CX	91		1	--------
XCHG AX,DX	92		1	--------
XCHG AX,BX	93		1	--------
XCHG AX,SP	94		1	--------
XCHG AX,BP	95		1	--------
XCHG AX,SI	96		1	--------
XCHG AX,DI	97		1	--------
XCHG rb,rmb	86 mr d0 d1	B	2~4	--------
XCHG rmb,rb	86 mr d0 d1	B	2~4	--------
XCHG rmw,rw	87 mr d0 d1	W	2~4	--------
XCHG rw,rmw	87 mr d0 d1	W	2~4	--------

Usage

```
        XCHG    dest,src
```

Modifies flags

None

Exchanges contents of source and destination.

XLAT/XLATB - Translate

mnemonics	op xx xx xx xx xx	sw	len	flags
XLAT	D7		1	--------

Usage

```
XLAT    translation-table
XLATB   (masm 5.x)
```

Modifies flags

None

Replaces the byte in AL with byte from a user table addressed by BX. The original value of AL is the index into the translate table.

XOR - Exclusive OR

mnemonics	op xx xx xx xx xx	sw	len	flags
XOR AL,ib	34 i0	B	2	0---szap
XOR AX,iw	35 i0 i1	W	3	0---szap
XOR rb,rmb	32 mr d0 d1	B	2~4	0---szap
XOR rw,rmw	33 mr d0 d1	W	2~4	0---szap
XOR rmb,ib	80 /6 d0 d1 i0	NB	3~5	0---szap
XOR rmw,iw	81 /6 d0 d1 i0 i1	NW	4~6	0---szap
XOR rmw,ib	83 /6 d0 d1 i0	EW	3~5	0---szap
XOR rmb,rb	30 mr d0 d1	B	2~4	0---szap
XOR rmw,rw	31 mr d0 d1	W	2~4	0---szap

Usage

```
XOR     dest,src
```

Modifies flags

CF OF PF SF ZF (AF undefined)

Performs a bitwise exclusive OR of the operands and returns the result in the destination.