

registers to calculate a 32-bit effective address. Thus the first byte of the encoding must be the effective address-size prefix, 01100111 (67h).

The *opcode* byte is the same (83h) as for the 80286 example described in the “Interpreting Encodings” section.

The *mod-reg-r/m* byte must specify a based indexed operand with a scaling factor of two. This operand cannot be specified with a single byte, so the encoding must also use the SIB byte. The value 100 in the *r/m* field specifies an SIB byte. The *reg* field is 000, as shown in the encoding. The *mod* field is 10 for operands that have base and scaled index registers and a 32-bit displacement. The combined *mod*, *reg*, and *r/m* fields for the second byte are 10000100 (84h).

The SIB byte is next. The scaling factor is 2, so the *ss* field is 01. The index register is ECX, so the *index* field is 001. The base register is EAX, so the *base* field is 000. The SIB byte is 01001000 (48h).

The next 4 bytes are the offset of **warray**. The low bytes are stored first. For this example, assume that **warray** is located at offset 10EFh. This offset only requires 2 bytes, but 4 must be supplied because of the addressing mode. A 32-bit address can be safely used in 16-bit mode as long as the upper word is 0.

The last byte of the instruction is used to store the 8-bit immediate value -3 (FDh). The encoding is shown here in hexadecimal:

```
67 83 84 48 00 00 EF 10 FD
```

Instructions

This section provides an alphabetical reference to the instructions for the 8086, 8088, 80286, 80386, and 80486 processors.

AAA ASCII Adjust After Addition

Adjusts the result of an addition to a decimal digit (0–9). The previous addition instruction should place its 8-bit sum in AL. If the sum is greater than 9h, AH is incremented and the carry and auxiliary carry flags are set. Otherwise, the carry and auxiliary carry flags are cleared.

Flags

```
O D I T S Z A P C
?           ? ? ± ? ±
```

Encoding	00110111			
	Syntax	Examples	CPU	Clock Cycles
	AAA	aaa	88/86	8
			286	3
			386	4
			486	3

AAD ASCII Adjust Before Division

Converts unpacked BCD digits in AH (most significant digit) and AL (least significant digit) to a binary number in AX. This instruction is often used to prepare an unpacked BCD number in AX for division by an unpacked BCD digit in an 8-bit register.

Flags O D I T S Z A P C
 ? ± ± ? ± ?

Encoding	11010101 00001010			
	Syntax	Examples	CPU	Clock Cycles
	AAD	aad	88/86	60
			286	14
			386	19
			486	14

AAM ASCII Adjust After Multiply

Converts an 8-bit binary number less than 100 decimal in AL to an unpacked BCD number in AX. The most significant digit goes in AH and the least significant in AL. This instruction is often used to adjust the product after a **MUL** instruction that multiplies unpacked BCD digits in AH and AL. It is also used to adjust the quotient after a **DIV** instruction that divides a binary number less than 100 decimal in AX by an unpacked BCD number.

Flags O D I T S Z A P C
? ± ± ? ± ?

Encoding 11010100 00001010

Syntax	Examples	CPU	Clock Cycles
AAM	aam	88/86	83
		286	16
		386	17
		486	15

AAS ASCII Adjust After Subtraction

Adjusts the result of a subtraction to a decimal digit (0–9). The previous subtraction instruction should place its 8-bit result in AL. If the result is greater than 9h, AH is decremented and the carry and auxiliary carry flags are set. Otherwise, the carry and auxiliary carry flags are cleared.

Flags O D I T S Z A P C
? ? ? ± ? ±

Encoding 00111111

Syntax	Examples	CPU	Clock Cycles
AAS	aas	88/86	8
		286	3
		386	4
		486	3

ADC Add with Carry

Adds the source operand, the destination operand, and the value of the carry flag. The result is assigned to the destination operand. This instruction is used to add the more significant portions of numbers that must be added in multiple registers.

Flags O D I T S Z A P C
± ± ± ± ± ± ±

Encoding 000100dw *mod,reg,r/m disp (0, 1, or 2)*

Syntax	Examples	CPU	Clock Cycles
ADC <i>reg,reg</i>	adc dx, cx	88/86	3
		286	2
		386	2
		486	1
ADC <i>mem,reg</i>	adc WORD PTR m32[2], dx	88/86	16+EA (W88=24+EA)
		286	7
		386	7
		486	3
ADC <i>reg,mem</i>	adc dx, WORD PTR m32[2]	88/86	9+EA (W88=13+EA)
		286	7
		386	6
		486	2

Encoding 100000sw *mod, 010,r/m disp (0, 1, or 2) data (1 or 2)*

Syntax	Examples	CPU	Clock Cycles
ADC <i>reg,immed</i>	adc dx, 12	88/86	4
		286	3
		386	2
		486	1
ADC <i>mem,immed</i>	adc WORD PTR m32[2], 16	88/86	17+EA (W88=23+EA)
		286	7
		386	7
		486	3

Encoding 0001010w *data (1 or 2)*

Syntax	Examples	CPU	Clock Cycles
ADC <i>accum,immed</i>	adc ax, 5	88/86	4
		286	3
		386	2
		486	1

ADD Add

Adds the source and destination operands and puts the sum in the destination operand.

Flags O D I T S Z A P C
 ± ± ± ± ± ±

Encoding 000000dw *mod,reg,r/m disp (0, 1, or 2)*

Syntax	Examples	CPU	Clock Cycles
ADD <i>reg,reg</i>	add ax, bx	88/86	3
		286	2
		386	2
		486	1
ADD <i>mem,reg</i>	add total, cx add array[bx+di], dx	88/86	16+EA (W88=24+EA)
		286	7
		386	7
		486	3
ADD <i>reg,mem</i>	add cx, incr add dx, [bp+6]	88/86	9+EA (W88=13+EA)
		286	7
		386	6
		486	2

Encoding 100000sw *mod, 000,r/m disp (p,1, or2) data (1or2)*

Syntax	Examples	CPU	Clock Cycles
ADD <i>reg,immed</i>	add bx, 6	88/86	4
		286	3
		386	2
		486	1
ADD <i>mem,immed</i>	add amount, 27 add pointers[bx][si], 6	88/86	17+EA (W88=23+EA)
		286	7
		386	7
		486	3

Encoding 0000010w *data (1 or 2)*

Syntax	Examples	CPU	Clock Cycles
ADD <i>accum,immed</i>	add ax, 10	88/86	4
		286	3
		386	2
		486	1

AND Logical AND

Performs a bitwise AND operation on the source and destination operands and stores the result in the destination operand. For each bit position in the operands, if both bits are set, the corresponding bit of the result is set. Otherwise, the corresponding bit of the result is cleared.

Flags O D I T S Z A P C
0 ± ± ? ± 0

Encoding 001000dw mod,reg,r/m disp (0, 1, or 2)

Syntax	Examples	CPU	Clock Cycles
AND <i>reg,reg</i>	and dx, bx	88/86	3
		286	2
		386	2
		486	1
AND <i>mem,reg</i>	and bitmask, bx and [bp+2], dx	88/86	16+EA (W88=24+EA)
		286	7
		386	7
		486	3
AND <i>reg,mem</i>	and bx, masker and dx, marray[bx+di]	88/86	9+EA (W88=13+EA)
		286	7
		386	6
		486	2

Encoding 100000sw mod, 100, r/m disp (0, 1, or 2) data (1 or 2)

Syntax	Examples	CPU	Clock Cycles
AND <i>reg,immed</i>	and dx, 0F7h	88/86	4
		286	3
		386	2
		486	1
AND <i>mem,immed</i>	and masker, 1001b	88/86	17+EA(W88=24+EA)
		286	7
		386	7
		486	3

Encoding 0010010w data (1 or 2)

Syntax	Examples	CPU	Clock Cycles
AND <i>accum,immed</i>	and ax, 0B6h	88/86	4
		286	3
		386	2
		486	1

ARPL Adjust Requested Privilege Level

80286–80486 Protected Only Verifies that the destination Requested Privilege Level (RPL) field (bits 0 and 1 of a selector value) is less than the source RPL field. If it is not, **ARPL** adjusts the destination RPL to match the source RPL. The destination operand should be a 16-bit memory or register operand containing the value of a selector. The source operand should be a 16-bit register containing the test value. The zero flag is set if the destination is adjusted; otherwise, the flag is cleared. **ARPL** is useful only in 80286–80486 protected mode. See Intel documentation for details on selectors and privilege levels.

Flags O D I T S Z A P C
±

Encoding 01100011 *mod,reg,r/m disp (0, 1, or 2)*

Syntax	Examples	CPU	Clock Cycles
ARPL <i>reg,reg</i>	arpl ax, cx	88/86 286 386 486	— 10 20 9
ARPL <i>mem,reg</i>	arpl selector, dx	88/86 286 386 486	— 11 21 9

BOUND Check Array Bounds

80286–80486 Only Verifies that a signed index value is within the bounds of an array. The destination operand can be any 16-bit register containing the index to be checked. The source operand must then be a 32-bit memory operand in which the low and high words contain the starting and ending values, respectively, of the array. (On the 80386–80486 processors, the destination operand can be a 32-bit register; in this case, the source operand must be a 64-bit operand made up of 32-bit bounds.) If the source operand is less than the first bound or greater than the last bound, an interrupt 5 is generated. The instruction pointer pushed by the interrupt (and returned by **IRET**) points to the **BOUND** instruction rather than to the next instruction.

Flags No change

Encoding 01100010 *mod, reg, r/m disp (2)*

Syntax	Examples	CPU	Clock Cycles
BOUND <i>reg16, mem32</i>	bound <i>di, base-4</i>	88/86	—
BOUND <i>reg32, mem64*</i>		286	<i>noj</i> =13†
		386	<i>noj</i> =10†
		486	<i>noj</i> =7

* 80386–80486 only.

† See **INT** for timings if interrupt 5 is called.

BSF/BSR Bit Scan

80386–80486 Only Scans an operand to find the first set bit. If a set bit is found, the zero flag is cleared and the destination operand is loaded with the bit index of the first set bit encountered. If no set bit is found, the zero flag is set. **BSF** (Bit Scan Forward) scans from bit 0 to the most significant bit. **BSR** (Bit Scan Reverse) scans from the most significant bit of an operand to bit 0.

Flags O D I T S Z A P C
±

Encoding 00001111 10111100 *mod, reg, r/m disp (0, 1, 2, or 4)*

Syntax	Examples	CPU	Clock Cycles
BSF <i>reg16, reg16</i>	bsf <i>cx, bx</i>	88/86	—
BSF <i>reg32, reg32</i>		286	—
		386	10+3 <i>n</i> *
		486	6–42†
BSF <i>reg16, mem16</i>	bsf <i>ecx, bitmask</i>	88/86	—
BSF <i>reg32, mem32</i>		286	—
		386	10+3 <i>n</i> *
		486	7–43§

Encoding 00001111 10111101 *mod, reg, r/m disp (0, 1, 2, or 4)*

Syntax	Examples	CPU	Clock Cycles
BSR <i>reg16,reg16</i>	bsr <i>cx, dx</i>	88/86	—
BSR <i>reg32,reg32</i>		286	—
		386	10+3 <i>n</i> *
		486	103 – 3 <i>n</i> #
BSR <i>reg16,mem16</i>	bsr <i>eax, bi tmask</i>	88/86	—
BSR <i>reg32,mem32</i>		286	—
		386	10+3 <i>n</i> *
		486	104 – 3 <i>n</i> #

* *n* = bit position from 0 to 31.

clocks = 6 if second operand equals 0.

† Clocks = 8 +

4 for each byte scanned +
 3 for each nibble scanned +
 3 for each bit scanned in last nibble
 or 6 if second operand equals 0.

§ Same as footnote above, but add 1 clock.

n = bit position from 0 to 31.

clocks = 7 if second operand equals 0.

BSWAP Byte Swap

80486 Only Takes a single 32-bit register as operand and exchanges the first byte with the fourth, and the second byte with the third. This instruction does not alter any bit values within the bytes and is useful for quickly translating between 8086-family byte storage and storage schemes in which the high byte is stored first.

Flags No change

Encoding 00001111 11001 *reg*

Syntax	Examples	CPU	Clock Cycles
BSWAP <i>reg32</i>	bswap <i>eax</i>	88/86	—
	bswap <i>ebx</i>	286	—
		386	—
		486	1

BT/BTC/BTR/BTS Bit Tests

80386–80486 Only Copies the value of a specified bit into the carry flag, where it can be tested by a **JC** or **JNC** instruction. The destination operand specifies the value in which the bit is located; the source operand specifies the bit position. **BT** simply copies the bit to the flag. **BTC** copies the bit and complements (toggles) it in the destination. **BTR** copies the bit and resets (clears) it in the destination. **BTS** copies the bit and sets it in the destination.

Flags O D I T S Z A P C
±

Encoding 00001111 10111010 *mod, BBB*,r/m disp (0, 1, 2, or 4) data (1)*

Syntax	Examples	CPU	Clock Cycles
BT <i>reg16,immed8</i> †	bt ax , 4	88/86 286 386 486	— — 3 3
BTC <i>reg16,immed8</i> †	bts ax , 4	88/86	—
BTR <i>reg16,immed8</i> †	btr bx , 17	286	—
BTS <i>reg16,immed8</i> †	btc edi , 4	386 486	6 6
BT <i>mem16,immed8</i> †	btr DWORD PTR [si], 27	88/86 286	— —
	btc col or[di] , 4	386 486	6 3
BTC <i>mem16,immed8</i> †	btc DWORD PTR	88/86	—
BTR <i>mem16,immed8</i> †	[bx], 27	286	—
BTS <i>mem16,immed8</i> †	btc maskit , 4	386	8
	btr col or[di] , 4	486	8

Encoding 00001111 10BBB011* *mod, reg, r/m disp (0, 1, 2, or 4)*

Syntax	Examples	CPU	Clock Cycles
BT <i>reg16,reg16</i> †	bt ax , bx	88/86 286 386 486	— — 3 3
BTC <i>reg16,reg16</i> †	btc eax , ebx	88/86	—
BTR <i>reg16,reg16</i> †	bts bx , ax	286	—
BTS <i>reg16,reg16</i> †	btr cx , di	386 486	6 6

Syntax	Examples	CPU	Clock Cycles
BT <i>mem16,reg16</i> [†]	bt [bx], dx	88/86	—
		286	—
		386	12
		486	8
BTC <i>mem16,reg16</i> [†]	bts flags[bx], cx	88/86	—
BTR <i>mem16,reg16</i> [†]	btr rotate, cx	286	—
BTS <i>mem16,reg16</i> [†]	btc [bp+8], si	386	13
		486	13

* *BBB* is 100 for **BT**, 111 for **BTC**, 110 for **BTR**, and 101 for **BTS**.

[†] Operands also can be 32 bits (*reg32* and *mem32*).

CALL Call Procedure

Calls a procedure. The instruction pushes the address of the next instruction onto the stack and jumps to the address specified by the operand. For **NEAR** calls, the offset (IP) is pushed and the new offset is loaded into IP.

For **FAR** calls, the segment (CS) is pushed and the new segment is loaded into CS. Then the offset (IP) is pushed and the new offset is loaded into IP. A subsequent **RET** instruction can pop the address so that execution continues with the instruction following the call.

Flags No change

Encoding 11101000 *disp* (2)

Syntax	Examples	CPU	Clock Cycles
CALL <i>label</i>	call upcase	88/86	19 (88=23)
		286	7+m
		386	7+m
		486	3

Encoding 10011010 *disp* (4)

Syntax	Examples	CPU	Clock Cycles
CALL <i>label</i>	call FAR PTR job call di stant	88/86	28 (88=36)
		286	13+m,pm=26+m*
		386	17+m,pm=34+m*
		486	18,pm=20*

Encoding 11111111 *mod,010,r/m*

Syntax	Examples	CPU	Clock Cycles
CALL <i>reg</i>	call <i>ax</i>	88/86	16 (88=20)
		286	7+m
		386	7+m
		486	5
CALL <i>mem16</i>	call <i>pointer</i>	88/86	21+EA (88=29+EA)
CALL <i>mem32</i> †	call [bx]	286	11+m
		386	10+m
		486	5

Encoding 11111111 *mod,011,r/m*

Syntax	Examples	CPU	Clock Cycles
CALL <i>mem32</i>	call <i>far_table[di]</i>	88/86	37+EA (88=53+EA)
CALL <i>mem48</i> †	call DWORD PTR [bx]	286	16+m, <i>pm</i> =29+m*
		386	22+m, <i>pm</i> =38+m*
		486	17, <i>pm</i> =20*

* Timings for calls through call and task gates are not shown, since they are used primarily in operating systems.

† 80386–80486 32-bit addressing mode only.

CBW Convert Byte to Word

Converts a signed byte in AL to a signed word in AX by extending the sign bit of AL into all bits of AH.

Flags No change

Encoding 10011000*

Syntax	Examples	CPU	Clock Cycles
CBW	cbw	88/86	2
		286	2
		386	3
		486	3

* **CBW** and **CWDE** have the same encoding with two exceptions: in 32-bit mode, **CBW** is preceded by the operand-size byte (66h) but **CWDE** is not; in 16-bit mode, **CWDE** is preceded by the operand-size byte but **CBW** is not.

CDQ Convert Double to Quad

80386–80486 Only Converts the signed doubleword in EAX to a signed quadword in the EDX:EAX register pair by extending the sign bit of EAX into all bits of EDX.

Flags No change

Encoding 10011001*

Syntax	Examples	CPU	Clock Cycles
CDQ	cdq	88/86 286 386 486	— — 2 3

* **CWD** and **CDQ** have the same encoding with two exceptions: in 32-bit mode, **CWD** is preceded by the operand-size byte (66h) but **CDQ** is not; in 16-bit mode, **CDQ** is preceded by the operand-size byte but **CWD** is not.

CLC Clear Carry Flag

Clears the carry flag.

Flags O D I T S Z A P C
0

Encoding 11111000

Syntax	Examples	CPU	Clock Cycles
CLC	cl c	88/86 286 386 486	2 2 2 2

CLD Clear Direction Flag

Clears the direction flag. All subsequent string instructions will process up (from low addresses to high addresses) by increasing the appropriate index registers.

Flags O D I T S Z A P C
0

Encoding 11111100

Syntax	Examples	CPU	Clock Cycles
CLD	<code>cld</code>	88/86 286 386 486	2 2 2 2

CLI Clear Interrupt Flag

Clears the interrupt flag. When the interrupt flag is cleared, maskable interrupts are not recognized until the flag is set again with the **STI** instruction. In protected mode, **CLI** clears the flag only if the current task's privilege level is less than or equal to the value of the IOPL flag. Otherwise, a general-protection fault occurs.

Flags O D I T S Z A P C
0

Encoding 11111010

Syntax	Examples	CPU	Clock Cycles
CLI	<code>cli</code>	88/86 286 386 486	2 3 3 5

CLTS Clear Task-Switched Flag

80286–80486 Privileged Only Clears the task-switched flag in the Machine Status Word (MSW) of the 80286, or the CR0 register of the 80386–80486. This instruction can be used only in system software executing at privilege level

0. See Intel documentation for details on the task-switched flag and other privileged-mode concepts.

Flags No change
 Encoding 00001111 00000110

Syntax	Examples	CPU	Clock Cycles
CLTS	clts	88/86 286 386 486	— 2 5 7

CMC Complement Carry Flag

Complements (toggles) the carry flag.

Flags O D I T S Z A P C
 ±
 Encoding 11110101

Syntax	Examples	CPU	Clock Cycles
CMC	cmc	88/86 286 386 486	2 2 2 2

CMP Compare Two Operands

Compares two operands as a test for a subsequent conditional-jump or set instruction. **CMP** does this by subtracting the source operand from the destination operand and setting the flags according to the result. **CMP** is the same as the **SUB** instruction, except that the result is not stored.

Flags O D I T S Z A P C
 ± ± ± ± ± ±

Encoding 001110dw mod, reg, r/m disp (0, 1, or 2)

Syntax	Examples	CPU	Clock Cycles
CMP <i>reg,reg</i>	cmp di, bx cmp dl, cl	88/86	3
		286	2
		386	2
		486	1
CMP <i>mem,reg</i>	cmp maximum, dx cmp array[si], bl	88/86	9+EA
		286	(W88=13+EA)
		386	7
		486	5
CMP <i>reg,mem</i>	cmp dx, minimum cmp bh, array[si]	88/86	9+EA
		286	(W88=13+EA)
		386	6
		486	6
			2

Encoding 100000sw mod, 111, r/m disp (0, 1, or 2) data (1 or 2)

Syntax	Examples	CPU	Clock Cycles
CMP <i>reg,immed</i>	cmp bx, 24	88/86	4
		286	3
		386	2
		486	1
CMP <i>mem,immed</i>	cmp WORD PTR [di], 4 cmp tester, 4000	88/86	10+EA
		286	(W88=14+EA)
		386	6
		486	5
			2

Encoding 0011110w data (1 or 2)

Syntax	Examples	CPU	Clock Cycles
CMP <i>accum,immed</i>	cmp ax, 1000	88/86	4
		286	3
		386	2
		486	1

CMPS/CMPSB/CMPSW/CMPSD Compare String

Compares two strings. DS:SI must point to the source string and ES:DI must point to the destination string (even if operands are given). For each comparison, the destination element is subtracted from the source element and the flags are updated to reflect the result (although the result is not stored). DI and SI are adjusted according to the size of the operands and the status of the direction flag. They are increased if the direction flag has been cleared with **CLD**, or decreased if the direction flag has been set with **STD**.

If the **CMPS** form of the instruction is used, operands must be provided to indicate the size of the data elements to be processed. A segment override can be given for the source (but not for the destination). If **CMPSB** (bytes), **CMPSW** (words), or **CMPSD** (doublewords on the 80386–80486 only) is used, the instruction determines the size of the data elements to be processed.

CMPS and its variations are normally used with repeat prefixes. **REPNE** (or **REPNZ**) is used to find the first match between two strings. **REPE** (or **REPZ**) is used to find the first mismatch. Before the comparison, CX should contain the maximum number of elements to compare. After a **REPNE CMPS**, the zero flag is clear if no match was found. After a **REPE CMPS**, the zero flag is set if no mismatch was found.

When the instruction finishes, ES:DI and DS:SI point to the element that follows (if the direction flag is clear) or precedes (if the direction flag is set) the match or mismatch. If CX decrements to 0, ES:DI and DS:SI point to the element that follows or precedes the last comparison. The zero flag is set or clear according to the result of the last comparison, not according to the value of CX.

Flags O D I T S Z A P C
 ± ± ± ± ± ±

Encoding 1010011 w

Syntax	Examples	CPU	Clock Cycles
CMPS [<i>segreg:</i>] <i>src</i> , [ES:] <i>dest</i>	cmps <i>source</i> , es: <i>dest</i>	88/86	22 ($W88=30$)
CMPSB [[<i>segreg:</i> [<i>src</i> ,] ES:] <i>dest</i>]	repne cmpsb	286	8
CMPSW [[<i>segreg:</i> [<i>src</i> ,] ES:] <i>dest</i>]	repe cmpsb	386	10
CMPSD [[<i>segreg:</i> [<i>src</i> ,] ES:] <i>dest</i>]	repne cmpsd	486	8

CMPXCHG Compare and Exchange

80486 Only Compares the destination operand to the accumulator (AL, AX, or EAX). If equal, the source operand is copied to the destination. Otherwise, the destination is copied to the accumulator. The instruction sets flags according to the result of the comparison.

Flags O D I T S Z A P C
± ± ± ± ± ± ± ±

Encoding 00001111 1011000b *mod, reg, r/m disp (0, 1, or 2)*

Syntax	Examples	CPU	Clock Cycles
CMPXCHG <i>mem,reg</i>	cmpxchg <i>warr[bx], cx</i>	88/86	—
	cmpxchg <i>string, bl</i>	286	—
		386	—
		486	7–10
CMPXCHG <i>reg,reg</i>	cmpxchg <i>dl, cl</i>	88/86	—
	cmpxchg <i>bx, dx</i>	286	—
		386	—
		486	6

CWD Convert Word to Double

Converts the signed word in AX to a signed doubleword in the DX:AX register pair by extending the sign bit of AX into all bits of DX.

Flags O D I T S Z A P C
± ± ± ± ± ± ± ±

Encoding 10011001*

Syntax	Examples	CPU	Clock Cycles
CWD	cwd	88/86	5
		286	2
		386	2
		486	3

* **CWD** and **CDQ** have the same encoding with two exceptions: in 32-bit mode, **CWD** is preceded by the operand-size byte (66h) but **CDQ** is not; in 16-bit mode, **CDQ** is preceded by the operand-size byte but **CWD** is not.

CWDE Convert Word to Extended Double

80386–80486 Only Converts a signed word in AX to a signed doubleword in EAX by extending the sign bit of AX into all bits of EAX.

Flags No change

Encoding 10011000*

Syntax	Examples	CPU	Clock Cycles
CWDE	<code>cwde</code>	88/86 286 386 486	— — 3 3

* **CBW** and **CWDE** have the same encoding with two exceptions: in 32-bit mode, **CBW** is preceded by the operand-size byte (66h) but **CWDE** is not; in 16-bit mode, **CWDE** is preceded by the operand-size byte but **CBW** is not.

DAA Decimal Adjust After Addition

Adjusts the result of an addition to a packed BCD number (less than 100 decimal). The previous addition instruction should place its 8-bit binary sum in AL. **DAA** converts this binary sum to packed BCD format with the least significant decimal digit in the lower four bits and the most significant digit in the upper four bits. If the sum is greater than 99h after adjustment, the carry and auxiliary carry flags are set. Otherwise, the carry and auxiliary carry flags are cleared.

Flags O D I T S Z A P C
? ± ± ± ± ±

Encoding 00100111

Syntax	Examples	CPU	Clock Cycles
DAA	<code>daa</code>	88/86 286 386 486	4 3 4 2

DAS Decimal Adjust After Subtraction

Adjusts the result of a subtraction to a packed BCD number (less than 100 decimal). The previous subtraction instruction should place its 8-bit binary result in AL. **DAS** converts this binary sum to packed BCD format with the least significant decimal digit in the lower four bits and the most significant digit in the upper four bits. If the sum is greater than 99h after adjustment, the carry and auxiliary carry flags are set. Otherwise, the carry and auxiliary carry flags are cleared.

Flags O D I T S Z A P C
? ± ± ± ± ±

Encoding 00101111

Syntax	Examples	CPU	Clock Cycles
DAS	das	88/86	4
		286	3
		386	4
		486	2

DEC Decrement

Subtracts 1 from the destination operand. Because the operand is treated as an unsigned integer, the **DEC** instruction does not affect the carry flag. To detect any effects on the carry flag, use the **SUB** instruction.

Flags O D I T S Z A P C
± ± ± ± ±

Encoding 1111111w *mod, 001, r/m disp (0, 1, or 2)*

Syntax	Examples	CPU	Clock Cycles
DEC <i>reg8</i>	dec <i>cl</i>	88/86	3
		286	2
		386	2
		486	1
DEC <i>mem</i>	dec <i>counter</i>	88/86	15+EA (W88=23+EA)
		286	7
		386	6
		486	3

Encoding	01001 <i>reg</i>			
	Syntax	Examples	CPU	Clock Cycles
	DEC <i>reg16</i>	dec ax	88/86	3
	DEC <i>reg32*</i>		286	2
			386	2
			486	1

* 80386–80486 only.

DIV Unsigned Divide

Divides an implied destination operand by a specified source operand. Both operands are treated as unsigned numbers. If the source (divisor) is 16 bits wide, the implied destination (dividend) is the DX:AX register pair. The quotient goes into AX and the remainder into DX. If the source is 8 bits wide, the implied destination operand is AX. The quotient goes into AL and the remainder into AH. On the 80386–80486, if the source is EAX, the quotient goes into EAX and the remainder into EDX.

Flags O D I T S Z A P C
? ? ? ? ? ? ?

Encoding 1111011*w mod, 110,r/m disp (0, 1, or 2)*

	Syntax	Examples	CPU	Clock Cycles
	DIV <i>reg</i>	di v cx di v dl	88/86	<i>b</i> =80–90, <i>w</i> =144–162
			286	<i>b</i> =14, <i>w</i> =22
			386	<i>b</i> =14, <i>w</i> =22, <i>d</i> =38
			486	<i>b</i> =16, <i>w</i> =24, <i>d</i> =40
	DIV <i>mem</i>	di v [bx] di v fsi ze	88/86	(<i>b</i> =86–96, <i>w</i> =150–168)+ <i>EA</i> *
			286	<i>b</i> =17, <i>w</i> =25
			386	<i>b</i> =17, <i>w</i> =25, <i>d</i> =41
			486	<i>b</i> =16, <i>w</i> =24, <i>d</i> =40

* Word memory operands on the 8088 take (158–176)+*EA* clocks.

ENTER Make Stack Frame

80286-80486 Only Creates a stack frame for a procedure that receives parameters passed on the stack. When *immed16* is 0, **ENTER** is equivalent to **push bp**, followed by **mov bp, sp**. The first operand of the **ENTER** instruction specifies the number of bytes to reserve for local variables. The second operand specifies the nesting level for the procedure. The nesting level should be 0 for languages that do not allow access to local variables of higher-level procedures (such as C, Basic, and FORTRAN). See the complementary instruction **LEAVE** for a method of exiting from a procedure.

Flags No change

Encoding 11001000 *data (2)* *data (1)*

Syntax	Examples	CPU	Clock Cycles
ENTER <i>immed16,0</i>	enter 4, 0	88/86	—
		286	11
		386	10
		486	14
ENTER <i>immed16,1</i>	enter 0, 1	88/86	—
		286	15
		386	12
		486	17
ENTER <i>immed16,immed8</i>	enter 6, 4	88/86	—
		286	$12+4(n-1)$
		386	$15+4(n-1)$
		486	$17+3n$

HLT Halt

Stops CPU execution until an interrupt restarts execution at the instruction following **HLT**. In protected mode, this instruction works only in privileged mode.

Flags No change

Encoding 11110100

Syntax	Examples	CPU	Clock Cycles
HLT	hl t	88/86	2
		286	2
		386	5
		486	4

IDIV Signed Divide

Divides an implied destination operand by a specified source operand. Both operands are treated as signed numbers. If the source (divisor) is 16 bits wide, the implied destination (dividend) is the DX:AX register pair. The quotient goes into AX and the remainder into DX. If the source is 8 bits wide, the implied destination is AX. The quotient goes into AL and the remainder into AH. On the 80386–80486, if the source is EAX, the quotient goes into EAX and the remainder into EDX.

Flags O D I T S Z A P C
 ? ? ? ? ? ?

Encoding 1111011w mod, 111,r/m disp (0, 1, or 2)

Syntax	Examples	CPU	Clock Cycles
IDIV <i>reg</i>	i di v bx i di v dl	88/86	<i>b</i> =101–112, <i>w</i> =165–184
		286	<i>b</i> =17, <i>w</i> =25
		386	<i>b</i> =19, <i>w</i> =27, <i>d</i> =43
		486	<i>b</i> =19, <i>w</i> =27, <i>d</i> =43
IDIV <i>mem</i>	i di v i temp	88/86	(<i>b</i> =107–118, <i>w</i> =171–190)+ <i>EA</i> *
		286	<i>b</i> =20, <i>w</i> =28
		386	<i>b</i> =22, <i>w</i> =30, <i>d</i> =46
		486	<i>b</i> =20, <i>w</i> =28, <i>d</i> =44

* Word memory operands on the 8088 take (175–194)+*EA* clocks.

IMUL Signed Multiply

Multiplies an implied destination operand by a specified source operand. Both operands are treated as signed numbers. If a single 16-bit operand is given, the implied destination is AX and the product goes into the DX:AX register pair. If a single 8-bit operand is given, the implied destination is AL and the product goes into AX. On the 80386–80486, if the operand is EAX, the product goes into the EDX:EAX register pair. The carry and overflow flags are set if the product is sign-extended into DX for 16-bit operands, into AH for 8-bit operands, or into EDX for 32-bit operands.

Two additional syntaxes are available on the 80186–80486 processors. In the two-operand form, a 16-bit register gives one of the factors and serves as the destination for the result; a source constant specifies the other factor. In the three-operand form, the first operand is a 16-bit register where the result will be stored, the second is a 16-bit register or memory operand containing one of the factors, and the third is a constant representing the other factor. With both variations, the overflow and carry flags are set if the result is too large to fit into the 16-bit destination register. Since the low 16 bits of the product are the same for both signed and unsigned multiplication, these syntaxes can be used for either signed or unsigned numbers. On the 80386–80486, the operands can be either 16 or 32 bits wide.

A fourth syntax is available on the 80386–80486. Both the source and destination operands can be given specifically. The source can be any 16- or 32-bit memory operand or general-purpose register. The destination can be any general-purpose register of the same size. The overflow and carry flags are set if the product does not fit in the destination.

Flags O D I T S Z A P C
± ? ? ? ? ±

Encoding 1111011w mod, 101, r/m disp (0, 1, or 2)

Syntax	Examples	CPU	Clock Cycles
IMUL <i>reg</i>	i mul dx	88/86 286 386 486	$b=80-98, w=128-154$ $b=13, w=21$ $b=9-14, w=9-22, d=9-38^*$ $b=13-18, w=13-26, d=13-42$
IMUL <i>mem</i>	i mul factor	88/86 286 386 486	$(b=86-104, w=134-160)+EA^\dagger$ $b=16, w=24$ $b=12-17, w=12-25, d=12-41^*$ $b=13-18, w=13-26, d=13-42$

* The 80386–80486 processors have an early-out multiplication algorithm. Therefore, multiplying an 8-bit or 16-bit value in EAX takes the same time as multiplying the value in AL or AX.

† Word memory operands on the 8088 take $(138-164)+EA$ clocks.

Encoding 011010s1 mod, reg, r/m disp (0, 1, or 2) data (1 or 2)

Syntax	Examples	CPU	Clock Cycles
IMUL <i>reg16, immed</i>	i mul cx, 25	88/86	—
IMUL <i>reg32, immed*</i>		286 386 486	21 $b=9-14, w=9-22, d=9-38^\dagger$ $b=13-18, w=13-26, d=13-42$

IMUL <i>reg16,reg16,immed</i>	i mul	88/86	—
IMUL <i>reg32,reg32,immed*</i>	dx, ax, 18	286	21
		386	<i>b=9-14,w=9-22,d=9-38†</i>
		486	<i>b=13-18,w=13-26,d=13-42</i>

Syntax	Examples	CPU	Clock Cycles
IMUL <i>reg16,mem16,immed</i>	i mul	88/86	—
IMUL <i>reg32,mem32,immed*</i>	bx, [si], 60	286	24
		386	<i>b=12–17,w=12–25,d=12–41</i> †
		486	<i>b=13–18,w=13–26,d=13–42</i>

Encoding 00001111 10101111 *mod,reg,r/m disp (0, 1, or 2)*

Syntax	Examples	CPU	Clock Cycles
IMUL <i>reg16,reg16</i>	i mul cx, ax	88/86	—
IMUL <i>reg32,reg32*</i>		286	—
		386	<i>w=9–22,d=9–38</i>
		486	<i>b=13–18,w=13–26,d=13–42</i>
IMUL <i>reg16,mem16</i>	i mul	88/86	—
IMUL <i>reg32,mem32*</i>	dx, [si]	286	—
		386	<i>w=12–25,d=12–41</i>
		486	<i>b=13–18,w=13–26,d=13–42</i>

* 80386–80486 only.

† The variations depend on the source constant size; destination size is not a factor.

IN Input from Port

Transfers a byte or word (or doubleword on the 80386–80486) from a port to the accumulator register. The port address is specified by the source operand, which can be DX or an 8-bit constant. Constants can be used only for port numbers less than 255; use DX for higher port numbers. In protected mode, a general-protection fault occurs if **IN** is used when the current privilege level is greater than the value of the IOPL flag.

Flags No change

Encoding 1110010w *data (1)*

Syntax	Examples	CPU	Clock Cycles
IN <i>accum,immed</i>	i n ax, 60h	88/86	10 (<i>W88=14</i>)
		286	5
		386	<i>12,pm=6,26*</i>
		486	<i>14,pm=9,29*†</i>

Encoding	1110110w			
	Syntax	Examples	CPU	Clock Cycles
	IN accum,DX	in ax, dx in al, dx	88/86 286 386 486	8 (W88=12) 5 13,pm=7,27* 14,pm=8,28*†

* First protected-mode timing: CPL ≤ IOPL. Second timing: CPL > IOPL.

† Takes 27 clocks in virtual 8086 mode.

INC Increment

Adds 1 to the destination operand. Because the operand is treated as an unsigned integer, the **INC** instruction does not affect the carry flag. If a signed carry requires detection, use the **ADD** instruction.

Flags	O D I T S Z A P C
	± ± ± ± ± ± ±

Encoding	1111111w mod,000,r/m disp (0, 1, or 2)			
	Syntax	Examples	CPU	Clock Cycles
	INC reg8	inc cl	88/86 286 386 486	3 2 2 1
	INC mem	inc vpage	88/86 286 386 486	15+EA (W88=23+EA) 7 6 3

Encoding	01000 reg			
	Syntax	Examples	CPU	Clock Cycles
	INC reg16 INC reg32*	inc bx	88/86 286 386 486	3 2 2 1

* 80386–80486 only.

INS/INSB/INSW/INSD Input from Port to String

80286-80486 Only Receives a string from a port. The string is considered the destination and must be pointed to by ES:DI (even if an operand is given). The input port is specified in DX. For each element received, DI is adjusted according to the size of the operand and the status of the direction flag. DI is increased if the direction flag has been cleared with **CLD** or decreased if the direction flag has been set with **STD**.

If the **INS** form of the instruction is used, a destination operand must be provided to indicate the size of the data elements to be processed, and DX must be specified as the source operand containing the port number. A segment override is not allowed. If **INSB** (bytes), **INSW** (words), or **INSD** (doublewords on the 80386–80486 only) is used, the instruction determines the size of the data elements to be received.

INS and its variations are normally used with the **REP** prefix. Before the repeated instruction is executed, CX should contain the number of elements to be received. In protected mode, a general-protection fault occurs if **INS** is used when the current privilege level is greater than the value of the IOPL flag.

Flags No change

Encoding 0110110_w

Syntax	Examples	CPU	Clock Cycles
INS [[ES:] <i>dest</i> , DX	<code>ins es: instr, dx</code>	88/86	—
INSB [[[ES:] <i>dest</i> , DX]	<code>rep insb</code>	286	5
INSW [[[ES:] <i>dest</i> , DX]	<code>rep insw</code>	386	15, <i>pm</i> =9,29*
INSD [[[ES:] <i>dest</i> , DX]	<code>rep insd</code>	486	17, <i>pm</i> =10,32*

* First protected-mode timing: CPL ≤ IOPL. Second timing: CPL > IOPL.

INT Interrupt

Generates a software interrupt. An 8-bit constant operand (0 to 255) specifies the interrupt procedure to be called. The call is made by indexing the interrupt number into the Interrupt Vector Table (IVT) starting at segment 0, offset 0. In real mode, the IVT contains 4-byte pointers to interrupt procedures. In privileged mode, the IVT contains 8-byte pointers.

When an interrupt is called in real mode, the flags, CS, and IP are pushed onto the stack (in that order), and the trap and interrupt flags are cleared. **STI** can be

used to restore interrupts. See Intel documentation and the documentation for your operating system for details on using and defining interrupts in privileged mode. To return from an interrupt, use the **IRET** instruction.

Flags O D I T S Z A P C
0 0

Encoding 11001101 *data (1)*

Syntax	Examples	CPU	Clock Cycles
INT <i>immed8</i>	int 25h	88/86	51 (88=71)
		286	23+m,pm=(40,78)+m*
		386	37,pm=59,99*
		486	30,pm=44,71*

Encoding 11001100

Syntax	Examples	CPU	Clock Cycles
INT 3	int 3	88/86	52 (88=72)
		286	23+m,pm=(40,78)+m*
		386	33,pm=59,99*
		486	26,pm=44,71*

* The first protected-mode timing is for interrupts to the same privilege level. The second is for interrupts to a higher privilege level. Timings for interrupts through task gates are not shown.

INTO Interrupt on Overflow

Generates Interrupt 4 if the overflow flag is set. The default MS-DOS behavior for Interrupt 4 is to return without taking any action. For **INTO** to have any effect, you must define an interrupt procedure for Interrupt 4.

Flags O D I T S Z A P C
± ±

Encoding 11001110

Syntax	Examples	CPU	Clock Cycles
INTO	into	88/86	53 (88=73),noj=4
		286	24+m,noj=3,pm=(40,78)+m*
		386	35,noj=3,pm=59,99*
		486	28,noj=3,pm=46,73*

* The first protected-mode timing is for interrupts to the same privilege level. The second is for interrupts to a higher privilege level. Timings for interrupts through task gates are not shown.

INVD Invalidate Data Cache

80486 Only Empties contents of the current data cache without writing changes to memory. Proper use of this instruction requires knowledge of how contents are placed in the cache. **INVD** is intended primarily for system programming. See Intel documentation for details.

Flags No change

Encoding 00001111 00001000

Syntax	Examples	CPU	Clock Cycles
INVD	<code>i nvd</code>	88/86 286 386 486	— — — 4

INVLPG Invalidate TLB Entry

80486 Only Invalidates an entry in the Translation Lookaside Buffer (TLB), used by the demand-paging mechanism in virtual-memory operating systems. The instruction takes a single memory operand and calculates the effective address of the operand, including the segment address. If the resulting address is mapped by any entry in the TLB, this entry is removed. Proper use of **INVLPG** requires understanding the hardware-supported demand-paging mechanism. **INVLPG** is intended primarily for system programming. See Intel documentation for details.

Flags No change

Encoding 00001111 00000001 *mod, reg, r/m disp (2)*

Syntax	Examples	CPU	Clock Cycles
INVLPG	<code>i nvl pg poi nter[bx]</code> <code>i nvl pg es: entry</code>	88/86 286 386 486	— — — 12*

* 11 clocks if address is not mapped by any TLB entry.

IRET/IRETD Interrupt Return

Returns control from an interrupt procedure to the interrupted code. In real mode, the **IRET** instruction pops IP, CS, and the flags (in that order) and resumes execution. See Intel documentation for details on **IRET** operation in privileged mode. On the 80386–80486, the **IRETD** instruction should be used to pop a 32-bit instruction pointer when returning from an interrupt called from a 32-bit segment. The **F** suffix prevents epilogue code from being generated when ending a **PROC** block. Use it to terminate interrupt service procedures.

Flags O D I T S Z A P C
± ± ± ± ± ± ± ± ±

Encoding 11001111

Syntax	Examples	CPU	Clock Cycles
IRET	<code>i ret</code>	88/86	32 (88=44)
IRETD*		286	17+m,pm=(31,55)+m†
IRETF		386	22,pm=38,82†
IRETDF*		486	15,pm=20,36

* 80386–80486 only.

† The first protected-mode timing is for interrupts to the same privilege level within a task. The second is for interrupts to a higher privilege level within a task. Timings for interrupts through task gates are not shown.

Jcondition Jump Conditionally

Transfers execution to the specified label if the flags condition is true. The *condition* is tested by checking the flags shown in the table on the following page. If *condition* is false, no jump is taken and program execution continues at the next instruction. On the 8086–80286 processors, the label given as the operand must be short (between –128 and +127 bytes from the instruction following the jump).* The 80386–80486 processors allow near jumps (–32,768 to +32,767 bytes). On the 80386–80486, the assembler generates the shortest jump possible, unless the jump size is explicitly specified.

When the 80386–80486 processors are in **FLAT** memory model, short jumps range from –128 to +127 bytes and near jumps range from –2 to +2 gigabytes. There are no far jumps.

Flags No change

Encoding	0111 <i>cond disp (1)</i>			
	Syntax	Examples	CPU	Clock Cycles
	<i>Jcondition label</i>	<code>jg bigger</code> <code>jo SHORT too_big</code> <code>jpe p_even</code>	88/86 286 386 486	16, <i>noj</i> =4 7+ <i>m,noj</i> =3 7+ <i>m,noj</i> =3 3, <i>noj</i> =1

Encoding	00001111 1000 <i>cond disp (2)</i>			
	Syntax	Examples	CPU	Clock Cycles
	<i>Jcondition label</i> †	<code>je next</code> <code>jnae lesser</code> <code>js negative</code>	88/86 286 386 486	— — 7+ <i>m,noj</i> =3 3, <i>noj</i> =1

* If a source file for an 8086–80286 program contains a conditional jump outside the range of –128 to +127 bytes, the assembler emits a level 3 warning and generates two instructions (including an unconditional jump) that are the equivalent of the desired instruction. This behavior can be enabled and disabled with the **OPTION LJMP** and **OPTION NOLJMP** directives.

† Near labels are only available on the 80386–80486. They are the default.

Jump Conditions	Opcode*	Mnemonic	Flags Checked	Description
<i>size</i> 0010	JB/JNAE	CF=1	Jump if below/not above or equal (unsigned comparisons)	
<i>size</i> 0011	JAE/JNB	CF=0	Jump if above or equal/not below (unsigned comparisons)	
<i>size</i> 0110	JBE/JNA	CF=1 or ZF=1	Jump if below or equal/not above (unsigned comparisons)	
<i>size</i> 0111	JA/JNBE	CF=0 and ZF=0	Jump if above/not below or equal (unsigned comparisons)	
<i>size</i> 0100	JE/JZ	ZF=1	Jump if equal (zero)	
<i>size</i> 0101	JNE/JNZ	ZF=0	Jump if not equal (not zero)	
<i>size</i> 1100	JL/JNGE	SF_OF	Jump if less/not greater or equal (signed comparisons)	
<i>size</i> 1101	JGE/JNL	SF=OF	Jump if greater or equal/not less (signed comparisons)	
<i>size</i> 1110	JLE/JNG	ZF=1 or SF_OF	Jump if less or equal/not greater (signed comparisons)	
<i>size</i> 1111	JG/JNLE	ZF=0 and SF=OF	Jump if greater/not less or equal (signed comparisons)	
<i>size</i> 1000	JS	SF=1	Jump if sign	
<i>size</i> 1001	JNS	SF=0	Jump if not sign	

Opcode*	Mnemonic	Flags Checked	Description
size 0010	JC	CF=1	Jump if carry
size 0011	JNC	CF=0	Jump if not carry
size 0000	JO	OF=1	Jump if overflow
size 0001	JNO	OF=0	Jump if not overflow
size 1010	JP/JPE	PF=1	Jump if parity/parity even
size 1011	JNP/JPO	PF=0	Jump if no parity/parity odd

* The *size* bits are 0111 for short jumps or 1000 for 80386–80486 near jumps.

JCXZ/JECXZ Jump if CX is Zero

Transfers program execution to the specified label if CX is 0. On the 80386–80486, **JECXZ** can be used to jump if ECX is 0. If the count register is not 0, execution continues at the next instruction. The label given as the operand must be short (between –128 and +127 bytes from the instruction following the jump).

Flags No change

Encoding 11100011 *disp (1)*

Syntax	Examples	CPU	Clock Cycles
JCXZ <i>label</i>	<code>j cxz not found</code>	88/86	18, <i>noj</i> =6
JECXZ <i>label</i> *		286	8+ <i>m</i> , <i>noj</i> =4
		386	9+ <i>m</i> , <i>noj</i> =5
		486	8, <i>noj</i> =5

* 80386–80486 only.

JMP Jump Unconditionally

Transfers program execution to the address specified by the destination operand. Jumps are near (between –32,768 and +32,767 bytes from the instruction following the jump), or short (between –128 and +127 bytes), or far (in a different code segment). Unless a distance is explicitly specified, the assembler selects the shortest possible jump. With near and short jumps, the operand specifies a new IP address. With far jumps, the operand specifies new IP and CS addresses.

When the 80386–80486 processors are in **FLAT** memory model, short jumps range from –128 to +127 bytes and near jumps range from –2 to +2 gigabytes.

Flags No change

Encoding 11101011 *disp (1)*

Syntax	Examples	CPU	Clock Cycles
JMP <i>label</i>	<code>j mp SHORT exit</code>	88/86 286 386 486	15 7+m 7+m 3

Encoding 11101001 *disp (2*)*

Syntax	Examples	CPU	Clock Cycles
JMP <i>label</i>	<code>j mp close</code> <code>j mp NEAR PTR distant</code>	88/86 286 386 486	15 7+m 7+m 3

Encoding 11101010 *disp (4*)*

Syntax	Examples	CPU	Clock Cycles
JMP <i>label</i>	<code>j mp FAR PTR close</code> <code>j mp distant</code>	88/86 286 386 486	15 11+m,pm=23+m† 12+m,pm=27+m† 17,pm=19†

Encoding 11111111 *mod,100,r/m disp (0 or 2)*

Syntax	Examples	CPU	Clock Cycles
JMP <i>reg16</i>	<code>j mp ax</code>	88/86	11
JMP <i>mem32§</i>		286 386 486	7+m 7+m 5
JMP <i>mem16</i>	<code>j mp WORD PTR [bx]</code>	88/86	18+EA
JMP <i>mem32§</i>	<code>j mp table[di]</code> <code>j mp DWORD PTR [si]</code>	286 386 486	11+m 10+m 5

Encoding 11111111 *mod,101,r/m disp (4*)*

Syntax	Examples	CPU	Clock Cycles
JMP <i>mem32</i>	jmp fpointer [<i>si</i>]	88/86	24+ <i>EA</i>
JMP <i>mem48</i> §	jmp DWORD PTR [<i>bx</i>]	286	15+ <i>m</i> , <i>pm</i> =26+ <i>m</i>
	jmp DWORD PTR [<i>di</i>]	386	12+ <i>m</i> , <i>pm</i> =27+ <i>m</i>
		486	13, <i>pm</i> =18

* On the 80386–80486, the displacement can be 4 bytes for near jumps or 6 bytes for far jumps.

† Timings for jumps through call or task gates are not shown, since they are normally used only in operating systems.

§ 80386–80486 only. You can use **DWORD PTR** to specify near register-indirect jumps or **DWORD PTR** to specify far register-indirect jumps.

LAHF Load Flags into AH Register

Transfers bits 0 to 7 of the flags register to AH. This includes the carry, parity, auxiliary carry, zero, and sign flags, but not the trap, interrupt, direction, or overflow flags.

Flags No change

Encoding 10011111

Syntax	Examples	CPU	Clock Cycles
LAHF	lahf	88/86	4
		286	2
		386	2
		486	3

LAR Load Access Rights

80286-80486 Protected Only Loads the access rights of a selector into a specified register. The source operand must be a register or memory operand containing a selector. The destination operand must be a register that will receive the access rights if the selector is valid and visible at the current privilege level. The zero flag is set if the access rights are transferred, or cleared if they are not. See Intel documentation for details on selectors, access rights, and other privileged-mode concepts.

Flags O D I T S Z A P C
±

Encoding 00001111 00000010 *mod, reg, r/m disp (0, 1, 2, or 4)*

Syntax	Examples	CPU	Clock Cycles
LAR <i>reg16,reg16</i>	lar <i>ax, bx</i>	88/86	—
LAR <i>reg32,reg32*</i>		286	14
		386	15
		486	11
LAR <i>reg16,mem16</i>	lar <i>cx, selector</i>	88/86	—
LAR <i>reg32,mem32*</i>		286	16
		386	16
		486	11

* 80386–80486 only.

LDS/LES/LFS/LGS/LSS Load Far Pointer

Reads and stores the far pointer specified by the source memory operand. The instruction moves the pointer's segment value into DS, ES, FS, GS, or SS (depending on the instruction). Then it moves the pointer's offset value into the destination operand. The **LDS** and **LES** instructions are available on all processors. The **LFS**, **LGS**, and **LSS** instructions are available only on the 80386–80486.

Flags No change

Encoding 11000101 *mod, reg, r/m disp (2)*

Syntax	Examples	CPU	Clock Cycles
LDS <i>reg,mem</i>	lds <i>si, fpointer</i>	88/86	16+EA (88=24+EA)
		286	7, <i>pm</i> =21
		386	7, <i>pm</i> =22
		486	6, <i>pm</i> =12

Encoding 11000100 *mod, reg, r/m disp (2)*

Syntax	Examples	CPU	Clock Cycles
LES <i>reg,mem</i>	les <i>di, fpointer</i>	88/86	16+EA (88=24+EA)
		286	7, <i>pm</i> =21
		386	7, <i>pm</i> =22
		486	6, <i>pm</i> =12

Encoding	00001111	10110100	<i>mod, reg, r/m</i>	<i>disp (2 or 4)</i>		
	Syntax		Examples		CPU	Clock Cycles
	LFS <i>reg,mem</i>		lfs <i>edi, fpointer</i>		88/86	—
					286	—
					386	7, <i>pm</i> =25
					486	6, <i>pm</i> =12

Encoding	00001111	10110101	<i>mod, reg, r/m</i>	<i>disp (2 or 4)</i>		
	Syntax		Examples		CPU	Clock Cycles
	LGS <i>reg,mem</i>		lgs <i>bx, fpointer</i>		88/86	—
					286	—
					386	7, <i>pm</i> =25
					486	6, <i>pm</i> =12

Encoding	00001111	10110010	<i>mod, reg, r/m</i>	<i>disp (2 or 4)</i>		
	Syntax		Examples		CPU	Clock Cycles
	LSS <i>reg,mem</i>		lss <i>bp, fpointer</i>		88/86	—
					286	—
					386	7, <i>pm</i> =22
					486	6, <i>pm</i> =12

LEA Load Effective Address

Calculates the effective address (offset) of the source memory operand and stores the result in the destination register. If the source operand is a direct memory address, the assembler encodes the instruction in the more efficient **MOV reg, immediate** form (equivalent to **MOV reg, OFFSET mem**).

Flags No change

Encoding	10001101	<i>mod, reg, r/m</i>	<i>disp (2)</i>		
	Syntax		Examples		CPU
	LEA <i>reg16,mem</i>		lea <i>bx, npointer</i>		88/86
	LEA <i>reg32,mem*</i>				286
					386
					486

* 80386–80486 only.

† 2 if index register used.

LEAVE High Level Procedure Exit

Terminates the stack frame of a procedure. **LEAVE** reverses the action of a previous **ENTER** instruction by restoring SP and BP to the values they had before the procedure stack frame was initialized. **LEAVE** is equivalent to **mov sp, bp**, followed by **pop bp**.

Flags No change

Encoding 11001001

Syntax	Examples	CPU	Clock Cycles
LEAVE	<code>leave</code>	88/86 286 386 486	— 5 4 5

LES/LFS/LGS Load Far Pointer to Extra Segment

See **LDS**.

LGDT/LIDT/LLDT Load Descriptor Table

Loads a value from an operand into a descriptor table register. **LGDT** loads into the Global Descriptor Table, **LIDT** into the Interrupt Vector Table, and **LLDT** into the Local Descriptor Table. These instructions are available only in privileged mode. See Intel documentation for details on descriptor tables and other protected-mode concepts.

Flags No change

Encoding 00001111 00000001 *mod, 010, r/m disp (2)*

Syntax	Examples	CPU	Clock Cycles
LGDT <i>mem48</i>	<code>lgdt descriptor</code>	88/86 286 386 486	— 11 11 11

Encoding	00001111	00000001	<i>mod, 011, r/m</i>	<i>disp (2)</i>		
	Syntax		Examples		CPU	Clock Cycles
	LIDT	<i>mem48</i>	l dt	descriptor	88/86	—
					286	12
					386	11
					486	11

Encoding	00001111	00000000	<i>mod, 010, r/m</i>	<i>disp (0, 1, or 2)</i>		
	Syntax		Examples		CPU	Clock Cycles
	LLDT	<i>reg16</i>	l1 dt	ax	88/86	—
					286	17
					386	20
					486	11
	LLDT	<i>mem16</i>	l1 dt	selector	88/86	—
					286	19
					386	24
					486	11

LMSW Load Machine Status Word

80286-80486 Privileged Only Loads a value from a memory operand into the Machine Status Word (MSW). This instruction is available only in privileged mode. See Intel documentation for details on the MSW and other protected-mode concepts.

Flags No change

Encoding	00001111	00000001	<i>mod, 110, r/m</i>	<i>disp (0, 1, or 2)</i>		
	Syntax		Examples		CPU	Clock Cycles
	LMSW	<i>reg16</i>	l msw	ax	88/86	—
					286	3
					386	10
					486	13
	LMSW	<i>mem16</i>	l msw	machine	88/86	—
					286	6
					386	13
					486	13

LOCK Lock the Bus

Locks out other processors during execution of the next instruction. This instruction is a prefix. It must precede an instruction that accesses a memory location that another processor might attempt to access at the same time. See Intel documentation for details on multiprocessor environments.

Flags No change

Encoding 11110000

Syntax	Examples	CPU	Clock Cycles
LOCK <i>instruction</i>	<code>lock xchg ax, sem</code>	88/86 286 386 486	2 0 0 1

LODS/LODSB/LODSW/LODSD Load Accumulator from String

Loads the accumulator register with an element from a string in memory. DS:SI must point to the source element, even if an operand is given. For each source element loaded, SI is adjusted according to the size of the operand and the status of the direction flag. SI is incremented if the direction flag has been cleared with **CLD** or decremented if the direction flag has been set with **STD**.

If the **LODS** form of the instruction is used, an operand must be provided to indicate the size of the data elements to be processed. A segment override can be given. If **LODSB** (bytes), **LODSW** (words), or **LODSD** (doublewords on the 80386–80486 only) is used, the instruction determines the size of the data elements to be processed and whether the element will be loaded to AL, AX, or EAX.

LODS and its variations are not used with repeat prefixes, since there is no reason to repeatedly load memory values to a register.

Flags No change

Encoding 1010110_w

Syntax	Examples	CPU	Clock Cycles
LODS [<i>segreg:</i>] <i>src</i>	<code>l ods es: source</code>	88/86	12 (<i>W88=16</i>)
LODSB [[<i>segreg:</i>] <i>src</i>]	<code>l odsw</code>	286	5
LODSW [[<i>segreg:</i>] <i>src</i>]		386	5
LODSD [[<i>segreg:</i>] <i>src</i>]		486	5

LOOP/LOOPW/LOOPD Loop

Loops repeatedly to a specified label. **LOOP** decrements CX (without changing any flags) and, if the result is not 0, transfers execution to the address specified by the operand. On the 80386–80486, **LOOP** uses the 16-bit CX in 16-bit mode and the 32-bit ECX in 32-bit mode. The default can be overridden with **LOOPW** (CX) or **LOOPD** (ECX). If CX is 0 after being decremented, execution continues at the next instruction. The operand must specify a short label (between –128 and +127 bytes from the instruction following the **LOOP** instruction).

Flags No change

Encoding 11100010 *disp (1)*

Syntax	Examples	CPU	Clock Cycles
LOOP <i>label</i>	<code>l oop wend</code>	88/86	17, <i>noj=5</i>
LOOPW <i>label*</i>		286	8+ <i>m,noj=4</i>
LOOPD <i>label*</i>		386	11+ <i>m</i>
		486	7, <i>noj=6</i>

* 80386–80486 only.

LOOPcondition/LOOPconditionW/LOOPconditionD Loop Conditionally

Loops repeatedly to a specified label if *condition* is met and if CX is not 0. On the 80386–80486, these instructions use the 16-bit CX in 16-bit mode and the 32-bit ECX in 32-bit mode. This default can be overridden with the **W** (CX) or **D** (ECX) forms of the instruction. The instruction decrements CX (without changing any flags) and tests whether the zero flag was set by a previous instruction (such as **CMP**). With **LOOPE** and **LOOPZ** (they are synonyms),

execution is transferred to the label if the zero flag is set and CX is not 0. With **LOOPNE** and **LOOPNZ** (they are synonyms), execution is transferred to the label if the zero flag is cleared and CX is not 0. Execution continues at the next instruction if the condition is not met. Before entering the loop, CX should be set to the maximum number of repetitions desired.

Flags No change

Encoding 11100001 *disp (1)*

Syntax	Examples	CPU	Clock Cycles
LOOPE <i>label</i>	<code>loopz again</code>	88/86	18, <i>noj</i> =6
LOOPEW <i>label*</i>		286	8+ <i>m</i> , <i>noj</i> =4
LOOPED <i>label*</i>		386	11+ <i>m</i>
LOOPZ <i>label</i>		486	9, <i>noj</i> =6
LOOPZW <i>label*</i>			
LOOPZD <i>label*</i>			

Encoding 11100000 *disp (1)*

Syntax	Examples	CPU	Clock Cycles
LOOPNE <i>label</i>	<code>loopnz for_next</code>	88/86	19, <i>noj</i> =5
LOOPNEW <i>label*</i>		286	8, <i>noj</i> =4
LOOPNED <i>label*</i>		386	11+ <i>m</i>
LOOPNZ <i>label</i>		486	9, <i>noj</i> =6
LOOPNZW <i>label*</i>			
LOOPNZD <i>label*</i>			

* 80386–80486 only.

LSL Load Segment Limit

80286-80486 Protected Only Loads the segment limit of a selector into a specified register. The source operand must be a register or memory operand containing a selector. The destination operand must be a register that will receive the segment limit if the selector is valid and visible at the current privilege level. The zero flag is set if the segment limit is transferred, or cleared if it is not. See Intel documentation for details on selectors, segment limits, and other protected-mode concepts.

Flags O D I T S Z A P C
±

Encoding 00001111 00000011 *mod, reg, r/m disp (0, 1, or 2)*

Syntax	Examples	CPU	Clock Cycles
LSL <i>reg16,reg16</i>	lsl ax, bx	88/86	—
LSL <i>reg32,reg32*</i>		286	14
		386	20,25†
		486	10
LSL <i>reg16,mem16</i>	lsl cx, seg_lim	88/86	—
LSL <i>reg32,mem32*</i>		286	16
		386	21,26†
		486	10

* 80386–80486 only.

† The first value is for byte granular; the second is for page granular.

LSS Load Far Pointer to Stack Segment

See **LDS**.

LTR Load Task Register

80286-80486 Protected Only Loads a value from the specified operand to the current task register. **LTR** is available only in privileged mode. See Intel documentation for details on task registers and other protected-mode concepts.

Flags No change

Encoding 00001111 00000000 *mod, 011,r/m disp (0, 1, or 2)*

Syntax	Examples	CPU	Clock Cycles
LTR <i>reg16</i>	ltr ax	88/86	—
		286	17
		386	23
		486	20
LTR <i>mem16</i>	ltr task	88/86	—
		286	19
		386	27
		486	20

MOV Move Data

Moves the value in the source operand to the destination operand. If the destination operand is SS, interrupts are disabled until the next instruction is executed (except on early versions of the 8088 and 8086).

Flags No change

Encoding 100010*d*w *mod, reg, r/m disp (0, 1, or 2)*

Syntax	Examples	CPU	Clock Cycles
MOV <i>reg,reg</i>	mov dh, bh	88/86	2
	mov dx, cx	286	2
	mov bp, sp	386	2
		486	1
MOV <i>mem,reg</i>	mov array[di], bx	88/86	9+EA (W88=13+EA)
	mov count, cx	286	3
		386	2
		486	1
MOV <i>reg,mem</i>	mov bx, pointer	88/86	8+EA (W88=12+EA)
	mov dx, matrix[bx+di]	286	5
		386	4
		486	1

Encoding 1100011*w mod, 000,r/m disp (0, 1, or 2) data (1 or 2)*

Syntax	Examples	CPU	Clock Cycles
MOV <i>mem,immed</i>	mov [bx], 15	88/86	10+EA (W88=14+EA)
	mov color, 7	286	3
		386	2
		486	1

Encoding 1011*w reg data (1 or 2)*

Syntax	Examples	CPU	Clock Cycles
MOV <i>reg,immed</i>	mov cx, 256	88/86	4
	mov dx, OFFSET string	286	2
		386	2
		486	1

Encoding 101000aw disp (2)

Syntax	Examples	CPU	Clock Cycles
MOV mem,accum	mov total, ax	88/86	10 (W88=14)
		286	3
		386	2
		486	1
MOV accum,mem	mov al, string	88/86	10 (W88=14)
		286	5
		386	4
		486	1

Encoding 100011d0 mod,sreg, r/m disp (0, 1, or 2)

Syntax	Examples	CPU	Clock Cycles
MOV sreg,reg16	mov ds, ax	88/86	2
		286	2,pm=17
		386	2,pm=18
		486	3,pm=9
MOV sreg,mem16	mov es, psp	88/86	8+EA (88=12+EA)
		286	5,pm=19
		386	5,pm=19
		486	3,pm=9
MOV reg16,sreg	mov ax, ds	88/86	2
		286	2
		386	2
		486	3
MOV mem16,sreg	mov stack_save, ss	88/86	9+EA (88=13+EA)
		286	3
		386	2
		486	3

MOV Move to/from Special Registers

80386–80486 Only Moves a value from a special register to or from a 32-bit general-purpose register. The special registers include the control registers CR0, CR2, and CR3; the debug registers DR0, DR1, DR2, DR3, DR6, and DR7; and the test registers TR6 and TR7. On the 80486, the test registers TR3, TR4, and TR5 are also available. See Intel documentation for details on special registers.

Flags

O	D	I	T	S	Z	A	P	C
?				?	?	?	?	?

Encoding	00001111 001000d0	11, <i>reg*</i> , <i>r/m</i>		
Syntax		Examples	CPU	Clock Cycles
MOV <i>reg32, controlreg</i>		mov <i>eax, cr2</i>	88/86 286 386 486	— — 6 4
MOV <i>controlreg, reg32</i>		mov <i>cr0, ebx</i>	88/86 286 386 486	— — CR0=10, CR2=4, CR3= 5 4, CR0=16

Encoding	00001111 001000d1	11, <i>reg*</i> , <i>r/m</i>		
Syntax		Examples	CPU	Clock Cycles
MOV <i>reg32, debugreg</i>		mov <i>edx, dr3</i>	88/86 286 386 486	— — DR0–3=22, DR6–7=14 10
MOV <i>debugreg, reg32</i>		mov <i>dr0, ecx</i>	88/86 286 386 486	— — DR0–3=22, DR6–7=16 11

Encoding	00001111 001001d0	11, <i>reg*</i> , <i>r/m</i>		
Syntax		Examples	CPU	Clock Cycles
MOV <i>reg32, testreg</i>		mov <i>edx, tr6</i>	88/86 286 386 486	— — 12 4, TR3=3
MOV <i>testreg, reg32</i>		mov <i>tr7, eax</i>	88/86 286 386 486	— — 12 4, TR3=6

* The *reg* field contains the register number of the special register (for example, 000 for CR0, 011 for DR7, or 111 for TR7).

MOVS/MOVS_B/MOVSW/MOVS_D Move String Data

Moves a string from one area of memory to another. DS:SI must point to the source string and ES:DI to the destination address, even if operands are given. For each element moved, DI and SI are adjusted according to the size of the operands and the status of the direction flag. They are increased if the direction flag has been cleared with **CLD**, or decreased if the direction flag has been set with **STD**.

If the **MOVS** form of the instruction is used, operands must be provided to indicate the size of the data elements to be processed. A segment override can be given for the source operand (but not for the destination). If **MOVS_B** (bytes), **MOVSW** (words), or **MOVS_D** (doublewords on the 80386–80486 only) is used, the instruction determines the size of the data elements to be processed.

MOVS and its variations are normally used with the **REP** prefix.

Flags No change

Encoding 1010010_w

Syntax	Examples	CPU	Clock Cycles
MOVS [[ES:] <i>dest</i> ,[[segreg:] <i>src</i>	rep movsb	88/86	18 (W88=26)
MOVS_B [[[ES:] <i>dest</i> ,[[segreg:] <i>src</i>	movs dest, es: source	286	5
MOVSW [[[ES:] <i>dest</i> ,[[segreg:] <i>src</i>		386	7
MOVS_D [[[ES:] <i>dest</i> ,[[segreg:] <i>src</i>		486	7

MOVX Move with Sign-Extend

80386–80486 Only Moves and sign-extends the value of the source operand to the destination register. **MOVX** is used to copy a signed 8-bit or 16-bit source operand to a larger 16-bit or 32-bit destination register.

Flags No change

Encoding 00001111 1011111_w *mod, reg, r/m disp (0, 1, 2, or 4)*

Syntax	Examples	CPU	Clock Cycles
MOVX <i>reg, reg</i>	movsx eax, bx	88/86	—
	movsx ecx, bl	286	—
	movsx bx, al	386	3
		486	3

Syntax	Examples	CPU	Clock Cycles
MOVZX <i>reg,mem</i>	movsx <i>cx, bsi gn</i>	88/86	—
	movsx <i>edx, wsi gn</i>	286	—
	movsx <i>eax, bsi gn</i>	386	6
		486	3

MOVZX Move with Zero-Extend

80386–80486 Only Moves and zero-extends the value of the source operand to the destination register. **MOVZX** is used to copy an unsigned 8-bit or 16-bit source operand to a larger 16-bit or 32-bit destination register.

Flags No change

Encoding 00001111 1011011w *mod, reg, r/m disp (0, 1, 2, or 4)*

Syntax	Examples	CPU	Clock Cycles
MOVZX <i>reg,reg</i>	movzx <i>eax, bx</i>	88/86	—
	movzx <i>ecx, bl</i>	286	—
	movzx <i>bx, al</i>	386	3
		486	3
MOVZX <i>reg,mem</i>	movzx <i>cx, bunsign</i>	88/86	—
	movzx <i>edx, wunsign</i>	286	—
	movzx <i>eax, bunsign</i>	386	6
		486	3

MUL Unsigned Multiply

Multiplies an implied destination operand by a specified source operand. Both operands are treated as unsigned numbers. If a single 16-bit operand is given, the implied destination is AX and the product goes into the DX:AX register pair. If a single 8-bit operand is given, the implied destination is AL and the product goes into AX. On the 80386–80486, if the operand is EAX, the product goes into the EDX:EAX register pair. The carry and overflow flags are set if DX is not 0 for 16-bit operands or if AH is not 0 for 8-bit operands.

Flags O D I T S Z A P C
± ? ? ? ? ±

Encoding	1111011w <i>mod, 100, r/m disp (0, 1, or 2)</i>			
	Syntax	Examples	CPU	Clock Cycles
	MUL <i>reg</i>	mul bx	88/86	$b=70-77, w=118-133$
		mul di	286	$b=13, w=21$
			386	$b=9-14, w=9-22, d=9-38^*$
			486	$b=13-18, w=13-26, d=13-42$
	MUL <i>mem</i>	mul factor	88/86	$(b=76-83, w=124-139)+EA^\dagger$
		mul WORD PTR [bx]	286	$b=16, w=24$
			386	$b=12-17, w=12-25, d=12-41^*$
			486	$b=13-18, w=13-26, d=13-42$

* The 80386–80486 processors have an early-out multiplication algorithm. Therefore, multiplying an 8-bit or 16-bit value in EAX takes the same time as multiplying the value in AL or AX.

† Word memory operands on the 8088 take $(128-143)+EA$ clocks.

NEG Two's Complement Negation

Replaces the operand with its two's complement. **NEG** does this by subtracting the operand from 0. If the operand is 0, the carry flag is cleared. Otherwise, the carry flag is set. If the operand contains the maximum possible negative value (-128 for 8-bit operands or $-32,768$ for 16-bit operands), the value does not change, but the overflow and carry flags are set.

Flags	O D I T S Z A P C
	± ± ± ± ± ± ±

Encoding	1111011w <i>mod, 011, r/m disp (0, 1, or 2)</i>			
	Syntax	Examples	CPU	Clock Cycles
	NEG <i>reg</i>	neg ax	88/86	3
			286	2
			386	2
			486	1
	NEG <i>mem</i>	neg balance	88/86	$16+EA$ ($W88=24+EA$)
			286	7
			386	6
			486	3

NOP No Operation

Performs no operation. **NOP** can be used for timing delays or alignment.

Flags No change

Encoding 10010000*

Syntax	Examples	CPU	Clock Cycles
NOP	nop	88/86	3
		286	3
		386	3
		486	3

* The encoding is the same as **XCHG AX,AX**.

NOT One's Complement Negation

Toggles each bit of the operand by clearing set bits and setting cleared bits.

Flags No change

Encoding 1111011w mod, 010, r/m disp (0,1,or2)

Syntax	Examples	CPU	Clock Cycles
NOT <i>reg</i>	not ax	88/86	3
		286	2
		386	2
		486	1
NOT <i>mem</i>	not masker	88/86	16+EA (W88=24+EA)
		286	7
		386	6
		486	3

OR Inclusive OR

Performs a bitwise OR operation on the source and destination operands and stores the result to the destination operand. For each bit position in the operands, if either or both bits are set, the corresponding bit of the result is set. Otherwise, the corresponding bit of the result is cleared.

Flags O D I T S Z A P C
0 ± ± ? ± 0

Encoding 000010dw mod, reg, r/m disp (0, 1, or 2)

Syntax	Examples	CPU	Clock Cycles
OR <i>reg,reg</i>	or ax, dx	88/86	3
		286	2
		386	2
		486	1
OR <i>mem,reg</i>	or bits, dx or [bp+6], cx	88/86	16+EA (W88=24+EA)
		286	7
		386	7
		486	3
OR <i>reg,mem</i>	or bx, masker or dx, color[di]	88/86	9+EA (W88=13+EA)
		286	7
		386	6
		486	2

Encoding 100000sw mod,001, r/m disp (0, 1, or 2) data (1 or 2)

Syntax	Examples	CPU	Clock Cycles
OR <i>reg,immed</i>	or dx, 110110b	88/86	4
		286	3
		386	2
		486	1
OR <i>mem,immed</i>	or flag_rec, 8	88/86	(b=17,w=25)+EA
		286	7
		386	7
		486	3

Encoding 0000110w data (1 or 2)

Syntax	Examples	CPU	Clock Cycles
OR <i>accum,immed</i>	or ax, 40h	88/86	4
		286	3
		386	2
		486	1

OUT Output to Port

Transfers a byte or word (or a doubleword on the 80386–80486) to a port from the accumulator register. The port address is specified by the destination operand, which can be **DX** or an 8-bit constant. In protected mode, a general-protection fault occurs if **OUT** is used when the current privilege level is greater than the value of the IOPL flag.

Flags No change

Encoding 1110011*w* *data (1)*

Syntax	Examples	CPU	Clock Cycles
OUT <i>immed8,accum</i>	out 60h, al	88/86 286 386 486	10 (88=14) 3 10, <i>pm</i> =4,24* 16, <i>pm</i> =11,31*

Encoding 1110111*w*

Syntax	Examples	CPU	Clock Cycles
OUT DX,accum	out dx, ax out dx, al	88/86 286 386 486	8 (88=12) 3 11, <i>pm</i> =5,25* 16, <i>pm</i> =10,30*

* First protected-mode timing: CPL < IOPL. Second timing: CPL > IOPL.

OUTS/OUTSB/OUTSW/OUTSD Output String to Port

80186–80486 Only Sends a string to a port. The string is considered the source and must be pointed to by DS:SI (even if an operand is given). The output port is specified in **DX**. For each element sent, **SI** is adjusted according to the size of the operand and the status of the direction flag. **SI** is increased if the direction flag has been cleared with **CLD**, or decreased if the direction flag has been set with **STD**.

If the **OUTS** form of the instruction is used, an operand must be provided to indicate the size of data elements to be sent. A segment override can be given. If **OUTSB** (bytes), **OUTSW** (words), or **OUTSD** (doublewords on the 80386–80486 only) is used, the instruction determines the size of the data elements to be sent.

OUTS and its variations are normally used with the **REP** prefix. Before the instruction is executed, **CX** should contain the number of elements to send. In protected mode, a general-protection fault occurs if **OUTS** is used when the current privilege level is greater than the value of the **IOPL** flag.

Flags No change

Encoding 0110111w

Syntax	Examples	CPU	Clock Cycles
OUTS <i>DX</i> , [<i>segreg:</i>] <i>src</i>	rep outs	88/86	—
OUTSB [DX , [<i>segreg:</i>] <i>src</i>]	dx , buffer	286	5
OUTSW [DX , [<i>segreg:</i>] <i>src</i>]	outsb	386	14, <i>pm</i> =8,28*
OUTSD [DX , [<i>segreg:</i>] <i>src</i>]	rep outsw	486	17, <i>pm</i> =10,32*

* First protected-mode timing: $CPL < IOPL$. Second timing: $CPL > IOPL$.

POP Pop

Pops the top of the stack into the destination operand. The value at **SS:SP** is copied to the destination operand and **SP** is increased by 2. The destination operand can be a memory location, a general-purpose 16-bit register, or any segment register except **CS**. Use **RET** to pop **CS**. On the 80386–80486, 32-bit values can be popped by giving a 32-bit operand. **ESP** is increased by 4 for 32-bit pops.

Flags No change

Encoding 01011 *reg*

Syntax	Examples	CPU	Clock Cycles
POP <i>reg16</i>	pop cx	88/86	8 (88=12)
POP <i>reg32*</i>		286	5
		386	4
		486	1

Encoding 10001111 *mod,000,r/m disp (2)*

Syntax	Examples	CPU	Clock Cycles
POP <i>mem16</i>	pop param	88/86	17+ <i>EA</i> (88=25+ <i>EA</i>)
POP <i>mem32*</i>		286	5
		386	5
		486	6

Encoding	000, <i>sreg</i> ,111			
	Syntax	Examples	CPU	Clock Cycles
	POP <i>segreg</i>	pop <i>es</i> pop <i>ds</i> pop <i>ss</i>	88/86 286 386 486	8 (88=12) 5, <i>pm</i> =20 7, <i>pm</i> =21 3, <i>pm</i> =9

Encoding	00001111 10, <i>sreg</i> ,001			
	Syntax	Examples	CPU	Clock Cycles
	POP <i>segreg</i> *	pop <i>fs</i> pop <i>gs</i>	88/86 286 386 486	— — 7, <i>pm</i> =21 3, <i>pm</i> =9

* 80386–80486 only.

POPA/POPAD Pop All

80186-80486 Only Pops the top 16 bytes on the stack into the eight general-purpose registers. The registers are popped in the following order: DI, SI, BP, SP, BX, DX, CX, AX. The value for the SP register is actually discarded rather than copied to SP. **POPA** always pops into 16-bit registers. On the 80386–80486, use **POPAD** to pop into 32-bit registers.

Flags No change

Encoding	01100001			
	Syntax	Examples	CPU	Clock Cycles
	POPA POPAD *	popa	88/86 286 386 486	— 19 24 9

* 80386–80486 only.

POPF/POPFD Pop Flags

Pops the value on the top of the stack into the flags register. **POPF** always pops into the 16-bit flags register. On the 80386–80486, use **POPFD** to pop into the 32-bit flags register.

Flags O D I T S Z A P C
 ± ± ± ± ± ± ± ± ±

Encoding 10011101

Syntax	Examples	CPU	Clock Cycles
POPF	<code>popf</code>	88/86	8 (88=12)
POPFD*		286	5
		386	5
		486	9, <i>pm</i> =6

* 80386–80486 only.

PUSH/PUSHW/PUSHD Push

Pushes the source operand onto the stack. SP is decreased by 2 and the source value is copied to SS:SP. The operand can be a memory location, a general-purpose 16-bit register, or a segment register. On the 80186–80486 processors, the operand can also be a constant. On the 80386–80486, 32-bit values can be pushed by specifying a 32-bit operand. ESP is decreased by 4 for 32-bit pushes. On the 8088 and 8086, **PUSH SP** saves the value of SP after the push. On the 80186–80486 processors, **PUSH SP** saves the value of SP before the push. The **PUSHW** and **PUSHD** instructions push a word (2 bytes) and a doubleword (4 bytes), respectively.

Flags No change

Encoding 01010 *reg*

Syntax	Examples	CPU	Clock Cycles
PUSH <i>reg16</i>	<code>push dx</code>	88/86	11 (88=15)
PUSH <i>reg32*</i>		286	3
PUSHW <i>reg16</i>		386	2
PUSHD <i>reg32*</i>		486	1

Encoding	11111111 <i>mod, 110,r/m disp (2)</i>			
	Syntax	Examples	CPU	Clock Cycles
	PUSH <i>mem16</i>	push [di]	88/86	16+EA (88=24+EA)
	PUSH <i>mem32*</i>	push fcount	286	5
			386	5
			486	4

Encoding	00, <i>sreg</i> ,110			
	Syntax	Examples	CPU	Clock Cycles
	PUSH <i>segreg</i>	push es	88/86	10 (88=14)
	PUSHW <i>segreg</i>	push ss	286	3
	PUSHD <i>segreg*</i>	push cs	386	2
			486	3

Encoding	00001111 10, <i>sreg</i> ,000			
	Syntax	Examples	CPU	Clock Cycles
	PUSH <i>segreg</i>	push fs	88/86	—
	PUSHW <i>segreg</i>	push gs	286	—
	PUSHD <i>segreg*</i>		386	2
			486	3

Encoding	011010s0 <i>data (1 or 2)</i>			
	Syntax	Examples	CPU	Clock Cycles
	PUSH <i>immed</i>	push 'a'	88/86	—
	PUSHW <i>immed</i>	push 15000	286	3
	PUSHD <i>immed*</i>		386	2
			486	1

* 80386–80486 only.

PUSHA/PUSHAD Push All

80186–80486 Only Pushes the eight general-purpose registers onto the stack. The registers are pushed in the following order: AX, CX, DX, BX, SP, BP, SI, DI. The value pushed for SP is the value before the instruction. **PUSHA** always pushes 16-bit registers. On the 80386–80486, use **PUSHAD** to push 32-bit registers.

Flags No change

Encoding 01100000

Syntax	Examples	CPU	Clock Cycles
PUSHA	pusha	88/86	—
PUSHAD*		286	17
		386	18
		486	11

* 80386–80486 only.

PUSHF/PUSHFD Push Flags

Pushes the flags register onto the stack. **PUSHF** always pushes the 16-bit flags register. On the 80386–80486, use **PUSHFD** to push the 32-bit flags register.

Flags No change

Encoding 10011100

Syntax	Examples	CPU	Clock Cycles
PUSHF	pushf	88/86	10(88=14)
PUSHFD*		286	3
		386	4
		486	4, <i>pm</i> =3

* 80386–80486 only.

RCL/RCR/ROL/ROR Rotate

Rotates the bits in the destination operand the number of times specified in the source operand. **RCL** and **ROL** rotate the bits left; **RCR** and **ROR** rotate right.

ROL and **ROR** rotate the number of bits in the operand. For each rotation, the leftmost or rightmost bit is copied to the carry flag as well as rotated. **RCL** and **RCR** rotate through the carry flag. The carry flag becomes an extension of the operand so that a 9-bit rotation is done for 8-bit operands, or a 17-bit rotation for 16-bit operands.

On the 8088 and 8086, the source operand can be either CL or 1. On the 80186–80486, the source operand can be CL or an 8-bit constant. On the 80186–80486, rotate counts larger than 31 are masked off, but on the 8088 and 8086, larger rotate counts are performed despite the inefficiency involved. The

overflow flag is modified only by single-bit variations of the instruction; for multiple-bit variations, the overflow flag is undefined.

Flags

O D I T S Z A P C
± ±

Encoding

1101000w mod, TTT*,r/m disp (0, 1, or 2)

Syntax	Examples	CPU	Clock Cycles
ROL <i>reg</i> ,1	ror ax , 1	88/86	2
ROR <i>reg</i> ,1	rol dl , 1	286	2
		386	3
		486	3
RCL <i>reg</i> ,1	rcl dx , 1	88/86	2
RCR <i>reg</i> ,1	rcr bl , 1	286	2
		386	9
		486	3
ROL <i>mem</i> ,1	ror bits , 1	88/86	15+EA (W88=23+EA)
ROR <i>mem</i> ,1	rol WORD PTR [bx] , 1	286	7
		386	7
		486	4
RCL <i>mem</i> ,1	rcl WORD PTR [si] , 1	88/86	15+EA (W88=23+EA)
RCR <i>mem</i> ,1	rcr WORD PTR m32[0] , 1	286	7
		386	10
		486	4

Encoding

1101001w mod, TTT*,r/m disp (0, 1, or 2)

Syntax	Examples	CPU	Clock Cycles
ROL <i>reg</i> ,CL	ror ax , cl	88/86	8+4n
ROR <i>reg</i> ,CL	rol dx , cl	286	5+n
		386	3
		486	3
RCL <i>reg</i> ,CL	rcl dx , cl	88/86	8+4n
RCR <i>reg</i> ,CL	rcr bl , cl	286	5+n
		386	9
		486	8–30
ROL <i>mem</i> ,CL	ror color , cl	88/86	20+EA+4n
ROR <i>mem</i> ,CL	rol WORD PTR [bp+6] , cl	286	(W88=28+EA+4n) 8+n
		386	7
		486	4

Syntax	Examples	CPU	Clock Cycles
RCL <i>mem,CL</i> RCR <i>mem,CL</i>	rcr WORD PTR [bx+di], cl rcl masker	88/86 286 386 486	20+EA+4n (W88=28+EA+4n) 8+n 10 9-31

Encoding

1100000w mod,TTT*,r/m disp (0, 1, or 2) data (1)

Syntax	Examples	CPU	Clock Cycles
ROL <i>reg,immed8</i> ROR <i>reg,immed8</i>	rol ax, 13 ror bl, 3 286	88/86 286 386 486	— 5+n 3 2
RCL <i>reg,immed8</i> RCR <i>reg,immed8</i>	rcl bx, 5 rcr si, 9	88/86 286 386 486	— 5+n 9 8-30
ROL <i>mem,immed8</i> ROR <i>mem,immed8</i>	rol BYTE PTR [bx], 10 ror bits, 6	88/86 286 386 486	— 8+n 7 4
RCL <i>mem,immed8</i> RCR <i>mem,immed8</i>	rcl WORD PTR [bp+8], rcr masker, 3	88/86 286 386 486	— 8+n 10 9-31

* TTT represents one of the following bit codes: 000 for **ROL**, 001 for **ROR**, 010 for **RCL**, or 011 for **RCR**.

REP Repeat String

Repeats a string instruction the number of times indicated by CX. First, CX is compared to 0; if it equals 0, execution proceeds to the next instruction. Otherwise, CX is decremented, the string instruction is performed, and the loop continues. **REP** is used with **MOVS** and **STOS**. **REP** also can be used with **INS** and **OUTS** on the 80186-80486 processors. On all processors except the 80386-80486, combining a repeat prefix with a segment override can cause errors if an interrupt occurs.

Flags No change

Encoding	11110011 1010010w			
	Syntax	Examples	CPU	Clock Cycles
	REP MOVS <i>dest,src</i>	rep movs <i>source, dest</i>	88/86	9+17 <i>n</i> (W88=9+25 <i>n</i>)
	REP MOVSB [[<i>dest,src</i>]]	rep movsb	286	5+4 <i>n</i>
	REP MOVSW [[<i>dest,src</i>]]		386	7+4 <i>n</i>
	REP MOVSD [[<i>dest,src</i>]]*		486	12+3 <i>n</i> #
Encoding	11110011 1010101w			
	Syntax	Examples	CPU	Clock Cycles
	REP STOS <i>dest</i>	rep stosb	88/86	9+10 <i>n</i> (W88=9+14 <i>n</i>)
	REP STOSB [[<i>dest</i>]]	rep stos <i>dest</i>	286	4+3 <i>n</i>
	REP STOSW [[<i>dest</i>]]		386	5+5 <i>n</i>
	REP STOSD [[<i>dest</i>]]*		486	7+4 <i>n</i> †
Encoding	11110011 1010101w			
	Syntax	Examples	CPU	Clock Cycles
	REP LODS <i>dest</i>	rep lodsb	88/86	—
	REP LODSB [[<i>dest</i>]]	rep lods <i>dest</i>	286	—
	REP LODSW [[<i>dest</i>]]		386	—
	REP LODSD [[<i>dest</i>]]*		486	7+4 <i>n</i> †
Encoding	11110011 0110110w			
	Syntax	Examples	CPU	Clock Cycles
	REP INS <i>dest,DX</i>	rep insb	88/86	—
	REP INSB [[<i>dest,DX</i>]]	rep ins <i>dest, dx</i>	286	5+4 <i>n</i>
	REP INSW [[<i>dest,DX</i>]]		386	13+6 <i>n</i> , <i>pm</i> =(7,27)+6 <i>n</i> §
	REP INSD [[<i>dest,DX</i>]]*		486	16+8 <i>n</i> , <i>pm</i> =(10,30)+8 <i>n</i> §
Encoding	11110011 0110111w			
	Syntax	Examples	CPU	Clock Cycles
	REP OUTS <i>DX,src</i>	rep outs <i>dx, source</i>	88/86	—
	REP OUTSB [[<i>src</i>]]	rep outsw	286	5+4 <i>n</i>
	REP OUTSW [[<i>src</i>]]		386	12+5 <i>n</i> , <i>pm</i> =(6,26)+5 <i>n</i> §
	REP OUTSD [[<i>src</i>]]*		486	17+5 <i>n</i> , <i>pm</i> =(11,31)+5 <i>n</i> §

* 80386–80486 only.

5 if *n* = 0, 13 if *n* = 1.

† 5 if *n* = 0.

§ First protected-mode timing: CPL ≤ IOPL. Second timing: CPL > IOPL.

REPcondition Repeat String Conditionally

Repeats a string instruction as long as *condition* is true and the maximum count has not been reached. **REPE** and **REPZ** (they are synonyms) repeat while the zero flag is set. **REPNE** and **REPNZ** (they are synonyms) repeat while the zero flag is cleared. The conditional-repeat prefixes should only be used with **SCAS** and **CMPS**, since these are the only string instructions that modify the zero flag. Before executing the instruction, CX should be set to the maximum allowable number of repetitions. First, CX is compared to 0; if it equals 0, execution proceeds to the next instruction. Otherwise, CX is decremented, the string instruction is performed, and the loop continues. On all processors except the 80386–80486, combining a repeat prefix with a segment override may cause errors if an interrupt occurs during a string operation.

Flags O D I T S Z A P C
±

Encoding 11110011 1010011w

Syntax	Examples	CPU	Clock Cycles
REPE CMPS <i>src,dest</i>	repz cmpsb	88/86	9+22 <i>n</i> (W88=9+30 <i>n</i>)
REPE CMPSB [<i>src,dest</i>]	repe cmps	286	5+9 <i>n</i>
REPE CMPSW [<i>src,dest</i>]	src, dest	386	5+9 <i>n</i>
REPE CMPSD [<i>src,dest</i>]*		486	7+7 <i>n</i> #

Encoding 11110011 1010111w

Syntax	Examples	CPU	Clock Cycles
REPE SCAS <i>dest</i>	repe scas dest	88/86	9+15 <i>n</i> (W88=9+19 <i>n</i>)
REPE SCASB [<i>dest</i>]	repz scasw	286	5+8 <i>n</i>
REPE SCASW [<i>dest</i>]		386	5+8 <i>n</i>
REPE SCASD [<i>dest</i>]*		486	7+5 <i>n</i> #

Encoding 11110010 1010011w

Syntax	Examples	CPU	Clock Cycles
REPNE CMPS <i>src,dest</i>	repne cmpsw	88/86	9+22 <i>n</i> (W88=9+30 <i>n</i>)
REPNE CMPSB [<i>src,dest</i>]	repnz cmps	286	5+9 <i>n</i>
REPNE CMPSW [<i>src,dest</i>]	src, dest	386	5+9 <i>n</i>
REPNE CMPSD [<i>src,dest</i>]*		486	7+7 <i>n</i> #

Encoding 11110010 1010111_w

Syntax	Examples	CPU	Clock Cycles
REPNE SCAS <i>des</i>	<code>repne scas dest</code>	88/86	9+15 <i>n</i> (<i>W88</i> =9+19 <i>n</i>)
REPNE SCASB [<i>dest</i>]	<code>repnz scasb</code>	286	5+8 <i>n</i>
REPNE SCASW [<i>dest</i>]		386	5+8 <i>n</i>
REPNE SCASD [<i>dest</i>]*		486	7+5 <i>n</i> *

* 80386–80486 only.

5 if *n*=0.

RET/RETN/RETF Return from Procedure

Returns from a procedure by transferring control to an address popped from the top of the stack. A constant operand can be given indicating the number of additional bytes to release. The constant is normally used to adjust the stack for arguments pushed before the procedure was called. The size of a return (near or far) is the size of the procedure in which the **RET** is defined with the **PROC** directive. **RETN** can be used to specify a near return; **RETF** can specify a far return. A near return pops a word into IP. A far return pops a word into IP and then pops a word into CS. After the return, the number of bytes given in the operand (if any) is added to SP.

Flags No change

Encoding 11000011

Syntax	Examples	CPU	Clock Cycles
RET	<code>ret</code>	88/86	16 (<i>88</i> =20)
RETN	<code>retn</code>	286	11+ <i>m</i>
		386	10+ <i>m</i>
		486	5

Encoding 11000010 *data* (2)

Syntax	Examples	CPU	Clock Cycles
RET <i>immed16</i>	<code>ret 2</code>	88/86	20 (<i>88</i> =24)
RETN <i>immed16</i>	<code>retn 8</code>	286	11+ <i>m</i>
		386	10+ <i>m</i>
		486	5

Encoding	11001011			
	Syntax	Examples	CPU	Clock Cycles
	RET	ret	88/86	26 (88=34)
	RETF	retf	286	15+m,pm=25+m,55*
			386	18+m,pm=32+m,62*
			486	13,pm=18,33*

Encoding	11001010 <i>data (2)</i>			
	Syntax	Examples	CPU	Clock Cycles
	RET <i>immed16</i>	ret 8	88/86	25 (88=33)
	RETF <i>immed16</i>	retf 32	286	15+m,pm=25+m,55*
			386	18+m,pm=32+m,68*
			486	14,pm=17,33*

* The first protected-mode timing is for a return to the same privilege level; the second is for a return to a lesser privilege level.

ROL/ROR Rotate

See **RCL/RCR**.

SAHF Store AH into Flags

Transfers AH into bits 0 to 7 of the flags register. This includes the carry, parity, auxiliary carry, zero, and sign flags, but not the trap, interrupt, direction, or overflow flags.

Flags O D I T S Z A P C
 ± ± ± ± ±

Encoding	10011110			
	Syntax	Examples	CPU	Clock Cycles
	SAHF	sahf	88/86	4
			286	2
			386	3
			486	2

SAL/SAR Shift

See **SHL/SHR/SAL/SAR**.

SBB Subtract with Borrow

Adds the carry flag to the second operand, then subtracts that value from the first operand. The result is assigned to the first operand. **SBB** is used to subtract the least significant portions of numbers that must be processed in multiple registers.

Flags O D I T S Z A P C
± ± ± ± ± ± ±

Encoding 000110dw *mod, reg, r/m disp (0, 1, or 2)*

Syntax	Examples	CPU	Clock Cycles
SBB <i>reg,reg</i>	sbb dx, cx	88/86	3
		286	2
		386	2
		486	1
SBB <i>mem,reg</i>	sbb WORD PTR m32[2], dx	88/86	16+EA (W88=24+EA)
		286	7
		386	6
		486	3
SBB <i>reg,mem</i>	sbb dx, WORD PTR m32[2]	88/86	9+EA (W88=13+EA)
		286	7
		386	7
		486	2

Encoding 100000sw *mod,011, r/m disp (0, 1, or 2) data (1 or 2)*

Syntax	Examples	CPU	Clock Cycles
SBB <i>reg,immed</i>	sbb dx, 45	88/86	4
		286	3
		386	2
		486	1
SBB <i>mem,immed</i>	sbb WORD PTR m32[2], 40	88/86	17+EA (W88=25+EA)
		286	7
		386	7
		486	3

Encoding	0001110w <i>data (1 or 2)</i>			
	Syntax	Examples	CPU	Clock Cycles
	SBB <i>accum,immed</i>	sbb ax, 320 88/86	4	
			86	3
			386	2
			486	1

SCAS/SCASB/SCASW/SCASD Scan String Flags

Scans a string to find a value specified in the accumulator register. The string to be scanned is considered the destination. ES:DI must point to that string, even if an operand is specified. For each element, the destination element is subtracted from the accumulator value and the flags are updated to reflect the result (although the result is not stored). DI is adjusted according to the size of the operands and the status of the direction flag. DI is increased if the direction flag has been cleared with **CLD**, or decreased if the direction flag has been set with **STD**.

If the **SCAS** form of the instruction is used, an operand must be provided to indicate the size of the data elements to be processed. No segment override is allowed. If **SCASB** (bytes), **SCASW** (words), or **SCASD** (doublewords on the 80386–80486 only) is used, the instruction determines the size of the data elements to be processed and whether the element scanned for is in AL, AX, or EAX.

SCAS and its variations are normally used with repeat prefixes. **REPNE** (or **REPNZ**) is used to find the first element in a string that matches the value in the accumulator register. **REPE** (or **REPZ**) is used to find the first mismatch. Before the scan, CX should contain the maximum number of elements to scan. After a **REPNE SCAS**, the zero flag is clear if the string does not contain the accumulator value. After a **REPE SCAS**, the zero flag is set if the string contains nothing but the accumulator value.

When the instruction finishes, ES:DI points to the element that follows (if the direction flag is clear) or precedes (if the direction flag is set) the match or mismatch. If CX decrements to 0, ES:DI points to the element that follows or precedes the last comparison. The zero flag is set or clear according to the result of the last comparison, not according to the value of CX.

Flags O D I T S Z A P C
 ± ± ± ± ± ±

Encoding 1010111_w

Syntax	Examples	CPU	Clock Cycles
SCAS [[ES:] <i>dest</i>	repne scasw	88/86	15 (W88=19)
SCASB [[[ES:] <i>dest</i>]	repe scasb	286	7
SCASW [[[ES:] <i>dest</i>]	scas es: <i>dest</i> i n	386	7
SCASD [[[ES:] <i>dest</i>]]*		486	6

* 80386–80486 only

SETcondition Set Conditionally

80386–80486 Only Sets the byte specified in the operand to 1 if *condition* is true or to 0 if *condition* is false. The condition is tested by checking the flags shown in the table on the following page. The instruction is used to set Boolean flags conditionally.

Flags No change

Encoding 00001111 1001*cond mod,000,r/m*

Syntax	Examples	CPU	Clock Cycles
SET <i>condition</i> <i>reg</i> 8	setc dh	88/86	—
	setz al	286	—
	setae bl	386	4
		486	true=4, false=3
SET <i>condition</i> <i>mem</i> 8	seto B <i>TYE</i> PTR [<i>ebx</i>]	88/86	—
	setle fl ag	286	—
	sete Bool eans[<i>di</i>]	386	5
		486	true=3, false=4

Set Conditions	Opcode	Mnemonic	Flags Checked	Description
	10010010	SETB/SETNAE	CF=1	Set if below/not above or equal (unsigned comparisons)
	10010011	SETAE/SETNB	CF=0	Set if above or equal/not below (unsigned comparisons)
	10010110	SETBE/SETNA	CF=1 or ZF=1	Set if below or equal/not above (unsigned comparisons)
	10010111	SETA/SETNBE	CF=0 and ZF=0	Set if above/not below or equal (unsigned comparisons)
	10010100	SETE/SETZ	ZF=1	Set if equal/zero
	10010101	SETNE/SETNZ	ZF=0	Set if not equal/not zero
	Opcode	Mnemonic	Flags Checked	Description

10011100	SETL/SETNGE	SF_OF	Set if less/not greater or equal (signed comparisons)
10011101	SETGE/SETNL	SF=OF	Set if greater or equal/not less (signed comparisons)
10011110	SETLE/SETNG	ZF=1 or SF_OF	Set if less or equal/not greater or equal (signed comparisons)
10011111	SETG/SETNLE	ZF=0 and SF=OF	Set if greater/not less or equal (signed comparisons)
10011000	SETS	SF=1	Set if sign
10011001	SETNS	SF=0	Set if not sign
10010010	SETC	F=1	Set if carry
10010011	SETNC	CF=0	Set if not carry
10010000	SETO	OF=1	Set if overflow
10010001	SETNO	OF=0	Set if not overflow
10011010	SETP/SETPE	PF=1	Set if parity/parity even
10011011	SETNP/SETPO	PF=0	Set if no parity/parity odd

SGDT/SIDT/SLDT Store Descriptor Table

80286-80486 Only Stores a descriptor table register into a specified operand. **SGDT** stores the Global Descriptor Table; **SIDT**, the Interrupt Vector Table; and **SLDT**, the Local Descriptor Table. These instructions are generally useful only in privileged mode. See Intel documentation for details on descriptor tables and other protected-mode concepts.

Flags No change

Encoding 00001111 00000001 *mod,000,r/m disp (2)*

Syntax	Examples	CPU	Clock Cycles
SGDT <i>mem48</i>	sgdt descriptor	88/86	—
		286	11
		386	9
		486	10

Encoding	00001111 00000001	<i>mod,001,r/m disp (2)</i>		
Syntax	Examples		CPU	Clock Cycles
SIDT <i>mem48</i>	sidt descriptor		88/86	—
			286	12
			386	9
			486	10
Encoding	00001111 00000000	<i>mod,000,r/m disp (0, 1, or 2)</i>		
Syntax	Examples		CPU	Clock Cycles
SLDT <i>reg16</i>	sldt ax		88/86	—
			286	2
			386	2
			486	2
SLDT <i>mem16</i>	sldt selector		88/86	—
			286	3
			386	2
			486	3

SHL/SHR/SAL/SAR Shift

Shifts the bits in the destination operand the number of times specified by the source operand. **SAL** and **SHL** shift the bits left; **SAR** and **SHR** shift right.

With **SHL**, **SAL**, and **SHR**, the bit shifted off the end of the operand is copied into the carry flag, and the leftmost or rightmost bit opened by the shift is set to 0. With **SAR**, the bit shifted off the end of the operand is copied into the carry flag, and the leftmost bit opened by the shift retains its previous value (thus preserving the sign of the operand). **SAL** and **SHL** are synonyms.

On the 8088 and 8086, the source operand can be either CL or 1. On the 80186–80486 processors, the source operand can be CL or an 8-bit constant. On the 80186–80486 processors, shift counts larger than 31 are masked off, but on the 8088 and 8086, larger shift counts are performed despite the inefficiency. Only single-bit variations of the instruction modify the overflow flag; for multiple-bit variations, the overflow flag is undefined.

Flags O D I T S Z A P C
 ± ± ± ? ± ±

Encoding	1101000w mod,TTT*,r/m disp (0, 1, or 2)		CPU	Clock Cycles
Syntax	Examples			
SAR <i>reg,1</i>	sar <i>di, 1</i> sar <i>cl, 1</i>	88/86	2	
		286	2	
		386	3	
		486	3	
SAL <i>reg,1</i>	shr <i>dh, 1</i>	88/86	2	
		286	2	
SHL <i>reg,1</i>	shl <i>si, 1</i>	386	3	
SHR <i>reg,1</i>	sal <i>bx, 1</i>	486	3	
SAR <i>mem,1</i>	sar <i>count, 1</i>	88/86	15+EA (W88=23+EA)	
		286	7	
		386	7	
		486	4	
SAL <i>mem,1</i>	sal WORD PTR <i>m32[0], 1</i>	88/86	15+EA (W88=23+EA)	
		286	7	
SHL <i>mem,1</i>	shl <i>index, 1</i>	386	7	
SHR <i>mem,1</i>	shr <i>unsign[di], 1</i>	486	4	

Encoding	1101001w mod,TTT*,r/m disp (0, 1, or 2)		CPU	Clock Cycles
Syntax	Examples			
SAR <i>reg,CL</i>	sar <i>bx, cl</i> sar <i>dx, cl</i>	88/86	8+4n	
		286	5+n	
		386	3	
		486	3	
SAL <i>reg,CL</i>	shr <i>dx, cl</i>	88/86	8+4n	
		286	5+n	
SHL <i>reg,CL</i>	shl <i>di, cl</i>	386	3	
SHR <i>reg,CL</i>	sal <i>ah, cl</i>	486	3	
SAR <i>mem,CL</i>	sar <i>sign, cl</i> sar WORD PTR [<i>bp+8</i>], <i>cl</i>	88/86	20+EA+4n (W88=28+EA+4n)	
		286	8+n	
		386	7	
		486	4	
SAL <i>mem,CL</i>	shr WORD PTR <i>m32[2], cl</i>	88/86	20+EA+4n (W88=28+EA+4n)	
		286	8+n	
SHL <i>mem,CL</i>	sal BYTE PTR [<i>di</i>], <i>cl</i>	386	7	
SHR <i>mem,CL</i>	shl <i>index, cl</i>	486	4	

Encoding 1100000w mod,TTT*,r/m disp (0, 1, or 2) data (1)

Syntax	Examples	CPU	Clock Cycles	
SAR <i>reg,immed8</i>	sar bx , 5	88/86	—	
	sar cl , 5	286	5+n	
		386	3	
		486	2	
SAL <i>reg,immed8</i>	sal cx , 6	88/86	—	
	SHL <i>reg,immed8</i>	shl di , 2	286	5+n
	SHR <i>reg,immed8</i>	shr bx , 8	386	3
		486	2	
SAR <i>mem,immed8</i>	sar sign_count , 3	88/86	—	
	sar WORD PTR [bx] , 5	286	8+n	
		386	7	
		486	4	
SAL <i>reg,immed8</i>	shr mem16 , 11	88/86	—	
SHL <i>reg,immed8</i>	shl unsign , 4	286	8+n	
SHR <i>reg,immed8</i>	sal array[bx+di] , 14	386	7	
		486	4	

* *TTT* represents one of the following bit codes: 100 for **SHL** or **SAL**, 101 for **SHR**, or 111 for **SAR**.

SHLD/SHRD Double Precision Shift

80386–80486 Only Shifts the bits of the second operand into the first operand. The number of bits shifted is specified by the third operand. **SHLD** shifts the first operand to the left by the number of positions specified in the count. The positions opened by the shift are filled by the most significant bits of the second operand. **SHRD** shifts the first operand to the right by the number of positions specified in the count. The positions opened by the shift are filled by the least significant bits of the second operand. The count operand can be either **CL** or an 8-bit constant. If a shift count larger than 31 is given, it is adjusted by using the remainder (modulo) of a division by 32.

Flags O D I T S Z A P C
? ± ± ? ± ±

Encoding	00001111 10100100	<i>mod,reg,r/m disp (0, 1, or 2) data (1)</i>		
	Syntax	Examples	CPU	Clock Cycles
	SHLD <i>reg16,reg16,immed8</i>	shl d ax, dx, 10	88/86	—
	SHLD <i>reg32,reg32,immed8</i>		286	—
			386	3
			486	2
	SHLD <i>mem16,reg16,immed8</i>	shl d bits, cx, 5	88/86	—
	SHLD <i>mem32,reg32,immed8</i>		286	—
			386	7
			486	3
Encoding	00001111 10101100	<i>mod,reg,r/m disp (0, 1, or 2) data (1)</i>		
	Syntax	Examples	CPU	Clock Cycles
	SHRD <i>reg16,reg16,immed8</i>	shrd cx, si, 3	88/86	—
	SHRD <i>reg32,reg32,immed8</i>		286	—
			386	3
			486	2
	SHRD <i>mem16,reg16,immed8</i>	shrd [di], dx, 13	88/86	—
	SHRD <i>mem32,reg32,immed8</i>		286	—
			386	7
			486	3
Encoding	00001111 10100101	<i>mod,reg,r/m disp (0, 1, or 2)</i>		
	Syntax	Examples	CPU	Clock Cycles
	SHLD <i>reg16,reg16,CL</i>	shl d ax, dx, cl	88/86	—
	SHLD <i>reg32,reg32,CL</i>		286	—
			386	3
			486	3
	SHLD <i>mem16,reg16,CL</i>	shl d	88/86	—
	SHLD <i>mem32,reg32,CL</i>	masker, ax, cl	286	—
			386	7
			486	4

Encoding	00001111 10101101	<i>mod,reg,r/m disp (0, 1, or 2)</i>		
Syntax	Examples	CPU	Clock Cycles	
SHRD <i>reg16,reg16,CL</i>	shrd bx, dx, cl	88/86	—	
SHRD <i>reg32,reg32,CL</i>		286	—	
		386	3	
		486	3	
SHRD <i>mem16,reg16,CL</i>	shrd [bx], dx, cl	88/86	—	
SHRD <i>mem32,reg32,CL</i>		286	—	
		386	7	
		486	4	

SMSGW Store Machine Status Word

80286-80486 Only Stores the Machine Status Word (MSW) into a specified memory operand. **SMSGW** is generally useful only in protected mode. See Intel documentation for details on the MSW and other protected-mode concepts.

Flags No change

Encoding	00001111 00000001	<i>mod,100,r/m disp (0, 1, or 2)</i>		
Syntax	Examples	CPU	Clock Cycles	
SMSGW <i>reg16</i>	smsgw ax	88/86	—	
		286	2	
		386	2	
		486	2	
SMSGW <i>mem16</i>	smsgw machine	88/86	—	
		286	3	
		386	3	
		486	3	

STC Set Carry Flag

Sets the carry flag.

Flags O D I T S Z A P C
1

Encoding 11111001

Syntax	Examples	CPU	Clock Cycles
STC	<code>stc</code>	88/86 286 386 486	2 2 2 2

STD Set Direction Flag

Sets the direction flag. All subsequent string instructions will process down (from high addresses to low addresses).

Flags O D I T S Z A P C
1

Encoding 11111101

Syntax	Examples	CPU	Clock Cycles
STD	<code>std</code>	88/86 286 386 486	2 2 2 2

STI Set Interrupt Flag

Sets the interrupt flag. When the interrupt flag is set, maskable interrupts are recognized. If interrupts were disabled by a previous **CLI** instruction, pending interrupts will not be executed immediately; they will be executed after the instruction following **STI**.

Flags O D I T S Z A P C
1

Encoding	11111011			
	Syntax	Examples	CPU	Clock Cycles
	STI	sti	88/86	2
			286	2
			386	3
			486	5

STOS/STOSB/STOSW/STOSD Store String Data

Stores the value of the accumulator in a string. The string is the destination and must be pointed to by ES:DI, even if an operand is given. For each source element loaded, DI is adjusted according to the size of the operand and the status of the direction flag. DI is incremented if the direction flag has been cleared with **CLD** or decremented if the direction flag has been set with **STD**.

If the **STOS** form of the instruction is used, an operand must be provided to indicate the size of the data elements to be processed. No segment override is allowed. If **STOSB** (bytes), **STOSW** (words), or **STOSD** (doublewords on the 80386–80486 only) is used, the instruction determines the size of the data elements to be processed and whether the element comes from AL, AX, or EAX.

STOS and its variations are often used with the **REP** prefix to fill a string with a repeated value. Before the repeated instruction is executed, CX should contain the number of elements to store.

Flags No change

Encoding 1010101w

Syntax	Examples	CPU	Clock Cycles
STOS [[ES:]] dest	stos es: dstring	88/86	11 (W88=15)
STOSB [[[ES:]] dest]	rep stosw	286	3
STOSW [[[ES:]] dest]	rep stosb	386	4
STOSD [[[ES:]] dest]*		486	5

* 80386–80486 only

STR Store Task Register

80286-80486 Only Stores the current task register to the specified operand. This instruction is generally useful only in privileged mode. See Intel documentation for details on task registers and other protected-mode concepts.

Flags No change

Encoding 00001111 00000000 *mod, 001, reg disp (0, 1, or 2)*

Syntax	Examples	CPU	Clock Cycles
STR <i>reg16</i>	str cx	88/86	—
		286	2
		386	2
		486	2
STR <i>mem16</i>	str taskreg	88/86	—
		286	3
		386	2
		486	3

SUB Subtract

Subtracts the source operand from the destination operand and stores the result in the destination operand.

Flags O D I T S Z A P C
± ± ± ± ± ± ±

Encoding 001010*dw* *mod, reg, r/m disp (0, 1, or 2)*

Syntax	Examples	CPU	Clock Cycles
SUB <i>reg,reg</i>	sub ax, bx sub bh, dh	88/86	3
		286	2
		386	2
		486	1
SUB <i>mem,reg</i>	sub tal1y, bx sub array[di], bl	88/86	16+EA (W88=24+EA)
		286	7
		386	6
		486	3

Syntax	Examples	CPU	Clock Cycles
SUB <i>reg,mem</i>	sub cx, di scard	88/86	9+EA (W88=13+EA)
	sub al, [bx]	286	7
		386	7
		486	2

Encoding 100000sw *mod,101,r/m disp (0, 1, or 2) data (1 or 2)*

Syntax	Examples	CPU	Clock Cycles
SUB <i>reg,immed</i>	sub dx, 45	88/86	4
	sub bl, 7	286	3
		386	2
		486	1
SUB <i>mem,immed</i>	sub total, 4000	88/86	17+EA (W88=25+EA)
	sub BYTE PTR [bx+di], 2	286	7
		386	7
		486	3

Encoding 0010110w *data (1 or 2)*

Syntax	Examples	CPU	Clock Cycles
SUB <i>accum,immed</i>	sub ax, 32000	88/86	4
		286	3
		386	2
		486	1

TEST Logical Compare

Tests specified bits of an operand and sets the flags for a subsequent conditional jump or set instruction. One of the operands contains the value to be tested. The other contains a bit mask indicating the bits to be tested. **TEST** works by doing a bitwise AND operation on the source and destination operands. The flags are modified according to the result, but the destination operand is not changed. This instruction is the same as the **AND** instruction, except the result is not stored.

Flags O D I T S Z A P C
0 ± ± ? ± 0

Encoding 1000010w *mod, reg, r/m disp (0, 1, or 2)*

Syntax	Examples	CPU	Clock Cycles
TEST <i>reg,reg</i>	test dx, bx	88/86	3
	test bl, ch	286	2
		386	2
		486	1
TEST <i>mem,reg</i>	test dx, flags	88/86	9+EA (W88=13+EA)
TEST <i>reg,mem*</i>	test bl, bitarray[bx]	286	6
		386	5
		486	2

Encoding 1111011w *mod,000,r/m disp (0, 1, or 2) data (1 or 2)*

Syntax	Examples	CPU	Clock Cycles
TEST <i>reg,immed</i>	test cx, 30h	88/86	5
	test cl, 1011b	286	3
		386	2
		486	1
TEST <i>mem,immed</i>	test masker , 1	88/86	11+EA
	test BYTE PTR [bx], 03h	286	6
		386	5
		486	2

Encoding 1010100w *data (1 or 2)*

Syntax	Examples	CPU	Clock Cycles
TEST <i>accum,immed</i>	test ax, 90h	88/86	4
		286	3
		386	2
		486	1

* MASM transposes **TEST** *reg,mem*; that is, it is encoded as **TEST** *mem,reg*.

VERR/VERW Verify Read or Write

80286-80486 Protected Only Verifies that a specified segment selector is valid and can be read or written to at the current privilege level. **VERR** verifies that the selector is readable. **VERW** verifies that the selector can be written to. If the segment is verified, the zero flag is set. Otherwise, the zero flag is cleared.

Flags O D I T S Z A P C
±

Encoding 00001111 00000000 *mod, 100,r/m disp (0, 1, or 2)*

Syntax	Examples	CPU	Clock Cycles
VERR <i>reg16</i>	verr ax	88/86	—
		286	14
		386	10
		486	11
VERR <i>mem16</i>	verr selector	88/86	—
		286	16
		386	11
		486	11

Encoding 00001111 00000000 *mod, 101,r/m disp (0, 1, or 2)*

Syntax	Examples	CPU	Clock Cycles
VERW <i>reg16</i>	verw cx	88/86	—
		286	14
		386	15
		486	11
VERW <i>mem16</i>	verw selector	88/86	—
		286	16
		386	16
		486	11

WAIT Wait

Suspends processor execution until the processor receives a signal that a coprocessor has finished a simultaneous operation. It should be used to prevent a coprocessor instruction from modifying a memory location that is being modified simultaneously by a processor instruction. **WAIT** is the same as the coprocessor **FWAIT** instruction.

Flags No change

Encoding 10011011

Syntax	Examples	CPU	Clock Cycles
WAIT	wai t	88/86	4
		286	3
		386	6
		486	1–3

WBINVD Write Back and Invalidate Data Cache

80486 Only Empties the contents of the current data cache after writing changes to memory. Proper use of this instruction requires knowledge of how contents are placed in the cache. **WBINVD** is intended primarily for system programming. See Intel documentation for details.

Flags No change

Encoding 00001111 00001001

Syntax	Examples	CPU	Clock Cycles
WBINVD	<code>wbi nvd</code>	88/86 286 386 486	— — — 5

XADD Exchange and Add

80486 Only Adds the source and destination operands and stores the sum in the destination; simultaneously, the original value of the destination is moved to the source. The instruction sets flags according to the result of the addition.

Flags O D I T S Z A P C
± ± ± ± ± ± ±

Encoding 00001111 1100000b *mod, reg, r/m disp (0, 1, or 2)*

Syntax	Examples	CPU	Clock Cycles
XADD <i>mem, reg</i>	<code>xadd warr[bx], ax</code>	88/86	—
	<code>xadd string, bl</code>	286	—
		386	—
		486	4
XADD <i>reg, reg</i>	<code>xadd dl, al</code>	88/86	—
	<code>xadd bx, dx</code>	286	—
		386	—
		486	3

XCHG Exchange

Exchanges the values of the source and destination operands.

Flags No change

Encoding 1000011w *mod,reg,r/m disp (0, 1, or 2)*

Syntax	Examples	CPU	Clock Cycles
XCHG <i>reg,reg</i>	xchg cx, dx	88/86	4
	xchg bl, dh	286	3
	xchg al, ah	386	3
		486	3
XCHG <i>reg,mem</i>	xchg [bx], ax	88/86	17+EA (W88=25+EA)
XCHG <i>mem,reg</i>	xchg bx, pointer	286	5
		386	5
		486	5

Encoding 10010 *reg*

Syntax	Examples	CPU	Clock Cycles
XCHG <i>accum,reg16*</i>	xchg ax, cx	88/86	3
XCHG <i>reg16,accum*</i>	xchg cx, ax	286	3
		386	3
		486	3

* On the 80386–80486, the accumulator may also be exchanged with a 32-bit register.

XLAT/XLATB Translate

Translates a value from one coding system to another by looking up the value to be translated in a table stored in memory. Before the instruction is executed, BX should point to a table in memory and AL should contain the unsigned position of the value to be translated from the table. After the instruction, AL contains the table value at the specified position. No operand is required, but one can be given to specify a segment override. DS is assumed unless a segment override is given. **XLATB** is a synonym for **XLAT**. Either version allows an operand, but neither requires one.

Flags No change

Encoding 11010111

Syntax	Examples	CPU	Clock Cycles
XLAT [[[<i>segreg</i> :] <i>mem</i>]]	<code>xl at</code>	88/86	11
XLATB [[[<i>segreg</i> :] <i>mem</i>]]	<code>xl atb es: table</code>	286	5
		386	5
		486	4

XOR Exclusive OR

Performs a bitwise exclusive OR operation on the source and destination operands and stores the result in the destination. For each bit position in the operands, if both bits are set or if both bits are cleared, the corresponding bit of the result is cleared. Otherwise, the corresponding bit of the result is set.

Flags O D I T S Z A P C
0 ± ± ? ± 0

Encoding 001100*dw mod, reg, r/m disp (0, 1, or 2)*

Syntax	Examples	CPU	Clock Cycles
XOR <i>reg, reg</i>	<code>xor cx, bx</code>	88/86	3
	<code>xor ah, al</code>	286	2
		386	2
		486	1
XOR <i>mem, reg</i>	<code>xor [bp+10], cx</code>	88/86	16+EA (W88=24+EA)
	<code>xor masked, bx</code>	286	7
		386	6
		486	3
XOR <i>reg, mem</i>	<code>xor cx, flags</code>	88/86	9+EA (W88=13+EA)
	<code>xor bl, bi tarray[di]</code>	286	7
		386	7
		486	2

Encoding	100000sw mod,110,r/m disp (0, 1, or 2) data (1 or 2)			
	Syntax	Examples	CPU	Clock Cycles
	XOR <i>reg,immed</i>	xor bx, 10h xor bl, 1	88/86 286 386 486	4 3 2 1
	XOR <i>mem,immed</i>	xor Boolean, 1 xor switches[bx], 101b	88/86 286 386 486	17+EA (W88=25+EA) 7 7 3

Encoding	0011010w data (1 or 2)			
	Syntax	Examples	CPU	Clock Cycles
	XOR <i>accum,immed</i>	xor ax, 01010101b	88/86 286 386 486	4 3 2 1