Section 10. 80286 Microprocessor Instruction Set

80286 Microprocessor Instruction Set

Data Transfer

MOV = move

Register to Register/Memory

1000100w	mod reg r/m

Register/Memory to Register

1000101w	mod reg r/m

Immediate to Register/Memory

1100011w	mod 000 r/m	data	data if w = 1

Immediate to Register

1011w reg	data	data if w = 1

Memory to Accumulator

1010000w	addr-low	addr-high

Accumulator to Memory

1010001w	addr-low	addr-high

Register/Memory to Segment Register

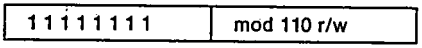
10001110	mod 0 reg r/m

Segment Register to Register/Memory

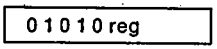
10001100	mod 0 reg r/m

PUSH = Push

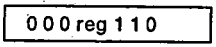
Memory



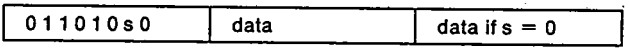
Register



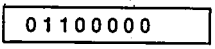
Segment Register



Immediate



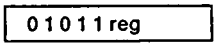
PUSHA = Push All



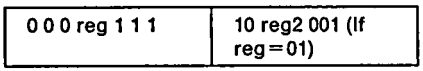
POP = Pop

Register/Memory

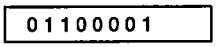
Register



Segment Register

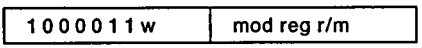


POPA = Pop All

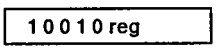


XCHG = Exchange

Register/Memory with Register



Register With Accumulator



IN = Input From

Fixed Port

1110010w	port number

Variable Port

1110110w

OUT = Output To

Fixed Port

1110011w	port number

Variable Port

1110111w

XLAT = Translate Byte to AL

11010111

LEA = Load EA to Register

10001101	mod reg r/m

LDS = Load Pointer to DS

11000101	mod reg r/m mod ≠ 11

LES = Load Pointer to ES

11000100	mod reg r/m mod ≠ 11

LAHF = Load AH into Flags

10011111

SAHF = Store AH Into Flags

10011110

PUSHF = Push Flags

10011100

POPF = Pop Flags

10011101

Arithmetic

ADD = Add

Register/Memory with Register to Either

000000dw	mod reg r/m

Immediate to Register/Memory

100000sw	mod 000 r/m	data	data if sw = 01

Immediate to Accumulator

0000010w	data	data if w = 1

ADC = Add with Carry

Register/Memory with Register to Either

000100dw	mod reg r/m

Immediate to Register/Memory

100000sw	mod 010 r/m	data	data if sw = 01

Immediate to Accumulator

0001010w	data	data if w = 1

INC = Increment

Register/Memory

1111111w	mod 000 r/m

Register

01000reg

SUB = Subtract

Register/Memory with Register to Either

0 0 1 0 1 0 dw	mod reg r/m

Immediate from Register/Memory

1 0 0 0 0 0 sw	mod 101 r/m	data	data if sw = 01

Immediate from Accumulator

0 0 1 0 1 1 0 w	data	data if w = 1

SBB = Subtract with Borrow

Register/Memory with Register to Either

0 0 0 1 1 0 dw	mod reg r/m

Immediate from Register/Memory

1 0 0 0 0 0 sw	mod 011 r/m	data	data if sw = 01

Immediate from Accumulator

0 0 0 1 1 1 0 w	data	data if w = 1

DEC = Decrement

Register/Memory

1 1 1 1 1 1 1 w	reg 001 r/m

Register

0 1 0 0 1 reg

CMP = Compare

Register/Memory with Register

0 0 1 1 1 0 dw	mod reg r/m

Register with Register/Memory

0 0 1 1 1 0 dw	mod reg r/m

Immediate with Register/Memory

1 0 0 0 0 0 sw	mod 111 r/m	data	data if sw = 01

Immediate with Accumulator

0 0 1 1 1 1 0 w	data	data if w = 1

NEG = Change Sign

1 1 1 1 0 1 1 w	mod 011 r/m

AAA = ASCII Adjust for Add

0 0 1 1 0 1 1 1

DAA = Decimal Adjust for Add

0 0 1 0 0 1 1 1

AAS = ASCII Adjust for Subtract

0 0 1 1 1 1 1 1

DAS = Decimal Adjust for Subtract

0 0 1 0 1 1 1 1

MUL = Multiply (Unsigned)

1 1 1 1 0 1 1 w	mod 100 r/m

IMUL = Integer Multiply (Signed)

1 1 1 1 0 1 1 w	mod 100 r/m

IIMUL = Integer Immediate Multiply (Signed)

0 1 1 0 1 0 s 1	mod reg r/m	data	data if s = 0

DIV = Divide (Unsigned)

1111011w	mod 110 r/m

IDIV = Integer Divide (Signed)

1111011w	mod 111 r/m

AAM = ASCII Adjust for Multiply

11010100	00001010

AAD = ASCII Adjust for Divide

11010101	00001010

CBW = Convert Byte to Word

10011000

CWD = Convert Word to Double Word

10011001

Logic

Shift/Rotate Instructions

Register/Memory by 1

1 1 0 1 0 0 0 w	mod TTT r/m

Register/Memory by CL

1 1 0 1 0 0 1 w	mod TTT r/m

Register/Memory by Count

1 1 0 0 0 0 0 w	mod TTT r/m	count

TTT	Instruction
0 0 0	ROL
0 0 1	ROR
0 1 0	RCL
0 1 1	RCR
1 0 0	SHL/SAL
1 0 1	SHR
1 1 1	SAR

AND = And

Register/Memory and Register to Either

0 0 1 0 0 0 d w	mod reg r/m

Immediate to Register/Memory

1 0 0 0 0 0 0 w	mod 100 r/m	data	data if w = 1

Immediate to Accumulator

0 0 1 0 0 1 0 w	data	data if w = 1

TEST = AND Function to Flags; No Result

Register/Memory and Register

1000010w	mod reg r/m

Immediate Data and Register/Memory

1111011w	mod 000 r/m	data	data if w = 1

Immediate Data and Accumulator

1010100w	data	data if w = 1

OR = Or

Register/Memory and Register to Either

000010dw	mod reg r/m

Immediate to Register/Memory

1000000w	mod 001 r/m	data	data if w = 1

Immediate to Accumulator

0000110w	data	data if w = 1

XOR = Exclusive OR

Register/Memory and Register to Either

001100dw	mod reg r/m

Immediate to Register/Memory

1000000w	mod 110 r/m	data	data if w = 1

Immediate to Accumulator

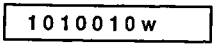
0011010w	data	data if w = 1

NOT = Invert Register/Memory

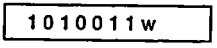
1111011w	mod 010 r/m

String Manipulation

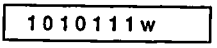
MOVS = Move Byte Word



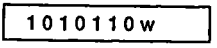
CMPS = Compare Byte Word



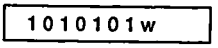
SCAS = Scan Byte Word



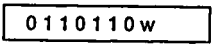
LODS = Load Byte/Word to AL/AX



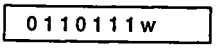
STOS = Store Byte/Word from AL/AX



INS = Input Byte/Word from DX Port



OUTS = Output Byte/Word to DX Port



Repeated String Manipulation

REP/REPNE, REPZ/REPNZ = Repeat String

Repeat Move String

11110011	1010010w

Repeat Compare String (z/Not z)

1111001z	1010011w

Repeat Scan String (z/Not z)

1111001z	1010111w

Repeat Load String

11110011	1010110w

Repeat Store String

11110011	1010101w

Repeat Input String

11110011	0110110w

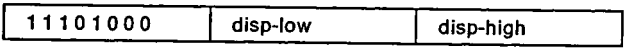
Repeat Output String

11110011	0110111w

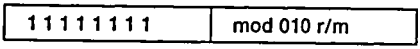
Control Transfer

CALL = Call

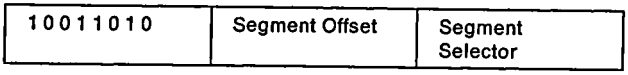
Direct Within Segment



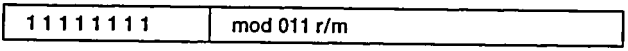
Register/Memory Indirect Within Segment



Direct Intersegment

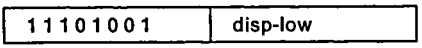


Indirect Intersegment

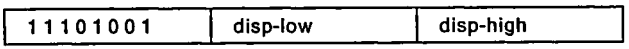


JMP = Unconditional Jump

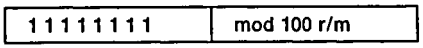
Short



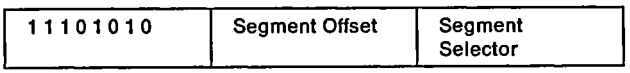
Direct within Segment



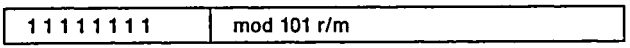
Register/Memory Indirect Within Segment



Direct Intersegment

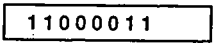


Indirect Intersegment

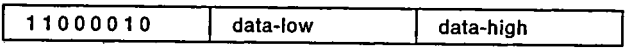


RET = Return from Call

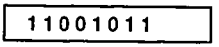
Within Segment



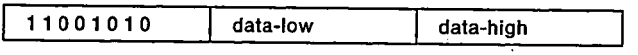
Within Segment Adding Immediate to SP



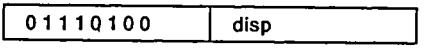
Intersegment



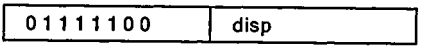
Intersegment Adding Immediate to SP



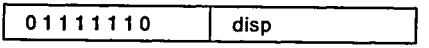
JE/JZ = Jump on Equal/Zero



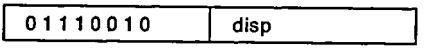
JL/JNGE = Jump on Less/Not Greater or Equal



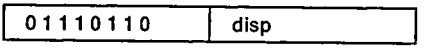
JLE/JNG = Jump on Less or Equal/Not Greater



JB/JNAE = Jump on Below/Not Above or Equal



JBE/JNA = Jump on Below or Equal/Not Above



JP/JPE = Jump on Parity/Parity Even

01111010	disp

JO = Jump on Overflow

01110000	disp

JS = Jump on Sign

01111000	disp

JNE/JNZ = Jump on Not Equal/Not Zero

01110101	disp

JNL/JGE = Jump on Not Less/Greater or Equal

01111101	disp

JNLE/JG = Jump on Not Less or Equal/Greater

01111111	disp

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

01110011	disp

JNBE/JA = Jump on Not Below or Equal/Above

01110111	disp

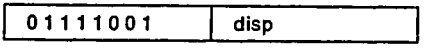
JNP/JPO = Jump on Not Parity/Parity Odd

01111011	disp

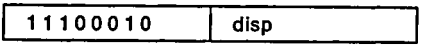
JNO = Jump on Not Overflow

01110001	disp

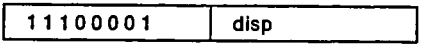
JNS = Jump on Not Sign



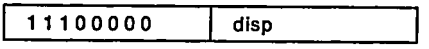
LOOP = Loop CX Times



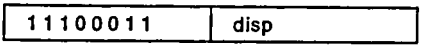
LOOPZ/LOOPE = Loop while Zero/Equal



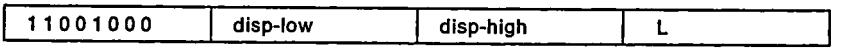
LOOPNZ/LOOPNE = Loop while Not Zero/Not Equal



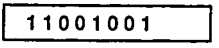
JCXZ = Jump on CX Zero



ENTER = Enter Procedure

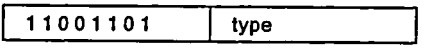


LEAVE = Leave Procedure

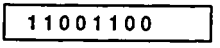


INT = Interrupt

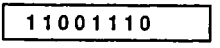
Type Specified



Type 3



INTO = Interrupt on Overflow



IRET = Interrupt Return

11001111

BOUND = Detect Value Out of Range

01100010	mod reg r/m

Processor Control

CLC = Clear Carry

11111000

CMC = Complement Carry

11110101

STC = Set Carry

11111001

CLD = Clear Direction

11111100

STD = Set Direction

11111101

CLI Clear Interrupt Enable Flag

11111010

STI = Set Interrupt Enable Flag

11111011

HLT = Halt

11110100

WAIT = Wait

10011011

LOCK = Bus Lock Prefix

11110000

CTS = Clear Task Switched Flag

00001111	00000110

ESC = Processor Extension Escape

11011TTT	mod LLL r/m

Protection Control

LGDT = Load Global Descriptor Table Register

00001111	00000001	mod 010 r/m

SGDT = Store Global Descriptor Table Register

00001111	00000001	mod 000 r/m

LIDT = Load Interrupt Descriptor Table Register

00001111	00000001	mod 011 r/m

SIDT = Store Interrupt Descriptor Table Register

00001111	00000001	mod 001 r/m

LLDT = Load Local Descriptor Table Register from Register/Memory

00001111	00000000	mod 010 r/m

SLDT = Store Local Descriptor Table Register to Register/Memory

00001111	00000000	mod 000 r/m

LTR = Load Task Register from Register/Memory

00001111	00000000	mod 011 r/m

STR = Store Task Register to Register/Memory

00001111	00000000	mod 001 r/m

LMSW = Load Machine Status Word from Register/Memory

00001111	00000001	mod 110 r/m

SMSW = Store Machine Status Word

00001111	00000001	mod 100 r/m

LAR = Load Access Rights from Register/Memory

00001111	00000010	mod reg r/m

LSL = Load Segment Limit from Register/Memory

00001111	00000011	mod reg r/m

ARPL = Adjust Requested Privilege Level from Register/Memory

01100011	mod reg r/m

VERR = Verify Read Access

00001111	00000000	mod 100 r/m

VERW = Verify Write Access

00001111	00000000	mod 101 r/m

The effective address (EA) of the memory operand is computed according to the mod and r/m fields:

If mod = 11, then r/m is treated as a reg field.
If mod = 00, then disp = 0, disp-low and disp-high are absent.
If mod = 01, then disp = disp-low sign-extended to 16 bits, disp-high is absent.
If mod = 10, then disp = disp-high:disp-low.

If r/m = 000, then EA = (BX) + (SI) + DISP
If r/m = 001, then EA = (BX) + (DI) + DISP
If r/m = 010, then EA = (BP) + (SI) + DISP
If r/m = 011, then EA = (BP) + (DI) + DISP
If r/m = 100, then EA = (SI) + DISP
If r/m = 101, then EA = (DI) + DISP
If r/m = 110, then EA = (BP) + DISP
If r/m = 111, then EA = (BX) + DISP.

DISP follows the second byte of the instruction (before data if required).

Note: An exception to the above statements occurs when mod=00 and r/m=110, in which case EA = disp-high; disp-low.

Segment Override Prefix

0 0 1 reg 1 1 0

The 2-bit and 3-bit reg fields are defined as follows:

Figure 10-1. 2-Bit Register Field

reg	Segment Register	reg	Segment Register
00	ES	10	SS
01	CS	11	DS

Figure 10-2. 3-Bit Register Field

16-bit (w = 1)	8-bit (w = 0)
000 AX	000 AL
001 CX	001 CL
010 DX	010 DL
011 BX	011 BL
100 SP	100 AH
101 BP	101 CH
110 SI	110 DH
111 DI	111 BH

The physical addresses of all operands addressed by the BP register are computed using the SS segment register. The physical addresses of the destination operands of the string primitive operations (those addressed by the DI register) are computed using the ES segment, which cannot be overridden.