Processor Operand-Size Prefix

The **operand-size prefix (66h)** is used to override the default segment attribute for the instruction it modifies.

This was necessary because the 8088/8086 instruction set used almost all 256 possible opcodes to handle instructions using 8- and 16-bit operands.

When Intel introduced 32-bit processors, they needed to find a way to invent new opcodes to handle 32-bit operands, yet retain compatibility with older processors.

For programs targeting 16-bit processors, a prefix byte was added to any instruction that used 32-bit operands.

For programs targeting 32-bit processors, 32-bit operands were the default, so a prefix byte was added to any instruction using 16-bit operands. Eight-bit operands need no prefix.

Here is an example of how to use the operand-size prefix in 16-bit mode:

```
715 .model small
716 .286
717 .stack 100h
718 .code
719 main PROC
720     mov ax, dx ; 8B C2
721     mov al, dl ; 8A C2
722 main ENDP
723 END main
```

The **mov ax,dx** instruction uses 16-bit operands, so it does not need a prefix byte.

The **mov al,dl** instruction uses 8-bit operands, so it also does not need a prefix byte.

Here is an example of how to use the operand-size prefix in 32-bit

mode:

```
.model small
.386
.stack 100h
.code
main PROC
    mov ax, dx ; 66 8B C2
    mov al, dl ; 8A C2
main ENDP
END main
```

The **mov ax,dx** instruction now needs a prefix byte to indicate that it is using 16-bit operands.

The **mov al,dl** instruction still does not need a prefix byte because it is using 8-bit operands.

The operand-size prefix is a powerful tool that allows programmers to control the operand size of instructions. This can be useful for improving performance or for ensuring compatibility with older processors.

MEMORY MODE INSTRUCTIONS

In x86 assembly language, the Memory Mode instructions are used to access memory operands with various addressing modes.

These instructions involve loading or storing data between registers and memory locations, with different combinations of registers and memory addresses.

The Mod field in the Mod R/M byte specifies the addressing mode for the memory operand.

Here are some key points about Mod Memory Mode instructions:

Mod R/M Byte: These instructions rely on the Mod R/M byte, which is a part of the instruction encoding, to specify the addressing mode and

operands. The Mod field (bits 6-7) in the Mod R/M byte defines the addressing mode.

Mod Field: The Mod field can take on various values:

• **00:** Register-indirect addressing mode. The memory operand is accessed via a register.

• **01:** Displacement-only addressing mode. An immediate value (displacement) is added to the base register.

• **10:** SIB (Scale-Index-Base) addressing mode. This is typically used for complex addressing calculations.

• **11:** Register mode. The operands are registers, not memory.

• **Registers and Memory:** Mod Memory Mode instructions can involve different combinations of registers and memory locations, making them versatile for reading from or writing to memory.

Code Explanation:

Now, let's go over the code provided in the earlier example:

```
770 main PROC
771     mov ax, [si]         ; Load AX with the value at memory address pointed to by SI
772     ; Mod 00, R/M 101 (SI is used as the offset address)
773
774     mov [si], al         ; Store the value in AL at the memory address pointed to by SI
775     ; Mod 00, R/M 101 (SI is used as the offset address)
776
777     add bx, 10h          ; Add 10h to BX
778     mov bx, [bx]         ; Load BX with the value at memory address [BX + 10h]
779     ; Mod 00, R/M 010 (Offset BX is used)
780
781     mov [bx + si], cx    ; Store the value in CX at memory address [BX + SI]
782     ; Mod 00, R/M 110 (Offset BX+SI is used)
```

mov ax, [si]: This instruction loads the value from the memory address pointed to by SI into the AX register. In the Mod R/M byte, it uses Mod 00 (register-indirect addressing mode) and R/M 101 (SI is the offset address).

mov [si], al: This instruction stores the value in AL into the memory address pointed to by SI. It also uses Mod 00 and R/M 101, as it's

effectively performing a register-indirect memory operation using SI as the offset.

add bx, 10h: This instruction adds 10h to the BX register.

mov bx, [bx]: This instruction loads the value from the memory address [BX + 10h] into the BX register. It uses Mod 00 (register-indirect addressing mode) and R/M 010 (BX is the base address with a displacement).

mov [bx + si], cx: This instruction stores the value in CX into the memory address [BX + SI]. It uses Mod 00 (register-indirect addressing mode) and R/M 110 (BX+SI is used as the offset).

These examples illustrate how to use Mod Memory Mode instructions to access memory operands using different addressing modes. The specific Mod R/M byte values indicate the addressing mode and operands used in each instruction.

REPEATING FOR SOME ADDITIONAL INFO:

The Mod R/M byte is a powerful tool that allows x86 assembly language programmers to specify a wide variety of memory-addressing modes.

The Mod field of the Mod R/M byte indicates the group of addressing modes, and the R/M field identifies the specific addressing mode within the group.

The following table shows the Mod R/M bytes for Mod 00:

R/M	Effective address
000	Register
001	[Base register]
010	[Base register + Displacement]
011	[Base register + Index register]
100	[Base register + Index register + Displacement]
101	[SI]
110	[DI]
111	[Base register + Displacement + 8]

The following examples show how to use the Mod R/M byte to encode MOV instructions that use different memory-addressing modes:

```
.model small
.data
    data1   dw 05h         ; Define a data item
    data2   dw 0A0Bh       ; Define another data item

.code
main PROC
    mov ax, [si]          ; Load AX with the value at memory address pointed to by SI
    ; Mod 00, R/M 101 (SI is used as the offset address)

    mov [si], al          ; Store the value in AL at the memory address pointed to by SI
    ; Mod 00, R/M 101 (SI is used as the offset address)

    add bx, 10h           ; Add 10h to BX
    mov bx, [bx]          ; Load BX with the value at memory address [BX + 10h]
    ; Mod 00, R/M 010 (Offset BX is used)

    mov [bx + si], cx     ; Store the value in CX at memory address [BX + SI]
    ; Mod 00, R/M 110 (Offset BX+SI is used)

    ; Terminate the program
    mov ah, 4Ch
    int 21h

main ENDP
END main
```

mov ax, [si]:

This instruction loads the value from the memory address pointed to by SI into the AX register.

Mod 00 indicates register-indirect addressing mode, and R/M 101 signifies SI as the offset address.

The value at memory address [SI] is read into AX.

mov [si], al:

This instruction stores the value in AL into the memory address pointed to by SI.

It uses Mod 00 (register-indirect addressing mode) and R/M 101 (SI is the offset address).

The content of AL is written to the memory location pointed to by SI.

add bx, 10h:

BX is loaded with the immediate value 10h.

The add instruction uses an immediate value to add 10h to the value in BX.

mov bx, [bx]:

This instruction loads the value from the memory address [BX + 10h] into the BX register.

Mod 00 signifies register-indirect addressing, and R/M 010 indicates the use of BX as the base address with a displacement of 10h.

BX now contains the value from memory address [BX + 10h].

mov [bx + si], cx:

This instruction stores the value in CX into the memory address [BX + SI].

Mod 00 indicates register-indirect addressing, and R/M 110 implies that both BX and SI are used as offsets.

The content of CX is written to the memory location at the address [BX + SI].

These examples illustrate how the Mod Memory Mode instructions enable the movement of data between registers and memory locations using different addressing modes.

The Mod R/M byte, as indicated in the comments, is key in specifying the addressing mode and operands for each instruction.

Use these tables as references when hand-assembling MOV instructions. (For more details, refer to the Intel manuals.)

Partial List of Mod R/M Bytes (16-Bit Segments).

Byte:		AL	CL	DL	BL	AH	CH	DH	BH	
Word:		AX	CX	DX	BX	SP	BP	SI	DI	
Register ID:		000	001	010	011	100	101	110	111	
Mod	R/M	Mod R/M Value								Effective Address
00	000	00	08	10	18	20	28	30	38	[BX + SI]
	001	01	09	11	19	21	29	31	39	[BX + DI]
	010	02	0A	12	1A	22	2A	32	3A	[BP + SI]
	011	03	0B	13	1B	23	2B	33	3B	[BP + DI]
	100	04	0C	14	1C	24	2C	34	3C	[SI]
	101	05	0D	15	1D	25	2D	35	3D	[DI]
	110	06	0E	16	1E	26	2E	36	3E	16-bit displacement
	111	07	0F	17	1F	27	2F	37	3F	[BX]

Table 2:

MOV Instruction Opcodes.

Opcode	Instruction	Description
88/r	MOV eb,rb	Move byte register into EA byte
89/r	MOV ew,rw	Move word register into EA word
8A/r	MOV rb,eb	Move EA byte into byte register
8B/r	MOV rw,ew	Move EA word into word register
8C/0	MOV ew,ES	Move ES into EA word
8C/1	MOV ew,CS	Move CS into EA word
8C/2	MOV ew,SS	Move SS into EA word
8C/3	MOV ew,DS	Move DS into EA word
8E/0	MOV ES,mw	Move memory word into ES
8E/0	MOV ES,rw	Move word register into ES
8E/2	MOV SS,mw	Move memory word into SS
8E/2	MOV SS,rw	Move register word into SS
8E/3	MOV DS,mw	Move memory word into DS
8E/3	MOV DS,rw	Move word register into DS
A0 dw	MOV AL,xb	Move byte variable (offset dw) into AL
A1 dw	MOV AX,xw	Move word variable (offset dw) into AX
A2 dw	MOV xb,AL	Move AL into byte variable (offset dw)
A3 dw	MOV xw,AX	Move AX into word register (offset dw)
B0 +rb db	MOV rb,db	Move immediate byte into byte register
B8 +rw dw	MOV rw,dw	Move immediate word into word register
C6 /0 db	MOV eb,db	Move immediate byte into EA byte
C7 /0 dw	MOV ew,dw	Move immediate word into EA word

Table 3:

Key to Instruction Opcodes.

/n:	A Mod R/M byte follows the opcode, possibly followed by immediate and displacement fields. The digit n (0–7) is the value of the reg field of the Mod R/M byte.
/r:	A Mod R/M byte follows the opcode, possibly followed by immediate and displacement fields.
db:	An immediate byte operand follows the opcode and Mod R/M bytes.
dw:	An immediate word operand follows the opcode and Mod R/M bytes.
+rb:	A register code (0–7) for an 8-bit register, which is added to the preceding hexadecimal byte to form an 8-bit opcode.
+rw:	A register code (0–7) for a 16-bit register, which is added to the preceding hexadecimal byte to form an 8-bit opcode.

Table 4:

Key to Instruction Operands.

db	A signed value between −128 and +127. If combined with a word operand, this value is sign-extended.
dw	An immediate word value that is an operand of the instruction.
eb	A byte-sized operand, either register or memory.
ew	A word-sized operand, either register or memory.
rb	An 8-bit register identified by the value (0–7).
rw	A 16-bit register identified by the value (0–7).
xb	A simple byte memory variable without a base or index register.
xw	A simple word memory variable without a base or index register.

Table 12-28 contains a few additional examples of MOV instructions that you can assemble by hand and compare to the machine code shown in the table. We assume that myWord begins at offset 0102h.

Sample MOV Instructions, with Machine Code.

Instruction	Machine Code	Addressing Mode
mov ax,myWord	A1 02 01	direct (optimized for AX)
mov myWord,bx	89 1E 02 01	direct
mov [di],bx	89 1D	indexed
mov [bx+2],ax	89 47 02	base-disp
mov [bx+si],ax	89 00	base-indexed
mov word ptr [bx+di+2],1234h	C7 41 02 34 12	base-indexed-disp

QUESTIONS

These questions are already answered, so just read:

Provide the opcodes for the following MOV instructions.

Instruction	Opcode
MOV AX, @DATA	A1
MOV DS, AX	8E D8
MOV AX, BX	89 D0
MOV BL, AL	88 C3
MOV AL, [SI]	8A 36
MOV MYBYTE, AL	88 44 01
MOV MYWORD, AX	89 84 01

Mod R/M bytes for MOV instructions:

Instruction	Mod R/M byte
MOV AX, @DATA	00 00 00
MOV DS, AX	8E D8
MOV DL, BL	00 10
MOV BL, [DI]	00 3E
MOV AX, [SI + 2]	04 36 02
MOV AX, ARRAY[SI]	00 46
MOV ARRAY[DI], AX	00 3C

Explanation

The Mod R/M byte is a complex topic, but it is essential to understand how to use it in order to write efficient and effective x86 assembly language code. The following is a brief explanation of how to calculate the Mod R/M byte for each of the MOV instructions in the question:

MOV AX, @DATA

The @DATA operand is a symbolic address that represents the global variable DATA. The Mod R/M byte for this instruction is 00 00 00. This is because the DATA operand is a 32-bit absolute address, and the Mod R/M byte for 32-bit absolute addresses is 00 00 00.

MOV DS, AX

The DS register is a segment register that stores the base address of the data segment. The Mod R/M byte for this instruction is 8E D8. This is because the DS register is a special register, and the Mod R/M byte for special registers is 8E. The D8 in the Mod R/M byte indicates that the destination operand is the DS register.

MOV AX, BX

The AX and BX registers are general-purpose registers. The Mod R/M byte for this instruction is 00 10. This is because the BX register is register number 3, and the Mod R/M byte for register-to-register moves is 00. The 10 in the Mod R/M byte indicates that the source operand is the BX register.

MOV BL, AL

The BL and AL registers are general-purpose registers. The Mod R/M byte for this instruction is 00 10. This is because the AL register is register number 0, and the Mod R/M byte for register-to-register moves is 00. The 10 in the Mod R/M byte indicates that the source operand is the AL register.

MOV AL, [SI]

The AL register is a general-purpose register, and the [SI] operand is a memory operand. The Mod R/M byte for this instruction is 00 36. This is because the SI register is register number 6, and the Mod R/M byte for memory operands is 00. The 36 in the Mod R/M byte indicates that the base register is the SI register.

MOV MYBYTE, AL

The MYBYTE operand is a global variable, and the AL register is a general-purpose register. The Mod R/M byte for this instruction is 88 44 01. This is because the MYBYTE operand is a memory operand, and the Mod R/M byte for memory operands is 00. The 44 in the Mod R/M byte indicates that the destination operand is the MYBYTE operand. The 01 in the Mod R/M byte indicates that the destination operand is a byte.

MOV MYWORD, AX

The MYWORD operand is a global variable, and the AX register is a general-purpose register. The Mod R/M byte for this instruction is 89 84 01. This is because the MYWORD operand is a memory operand, and the Mod