机器语言

多字节指令1到7个字节组成。

操作码OP表示指令所执行的操作。

OP d w

w位用来表示本指令是对字(w=1)还是对字节(w=0)进行操作。

d位在双操作数指令中才有效，d位指定寄存器用于目的操作数(d=1)还是源操作数(d=0)。

OPsw

8位立即数扩展为16位立即数时s=1。

mdregr/m

mod r/m register/memory

x86与x86-64指令集的指令的格式为：

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **指令前缀** | **指令码** | **ModR/M** | **SIB** | **偏移** | **直接数** |
| Instruction Prefixes | Opcode |  |  | Displacement | Immediate |
| 可选。 | 单字节、双字节、三字节 | 按需。 | 按需。 | 0、1、2、4字节长 | 0、1、2、4字节长 |
| 最多4个单字节前缀。 | 0-2位：R/M | 0-2位：Base |
| 任何顺序均可。 | 3-5位：Reg/Opcode | 3-5位：Index |
|  | 6-7位：Mod | 6-7位：Scale |

指令前缀

分为4组，每组用1个字节编码。每组在指令中至多指定1个前缀值。4组的顺序可以任意。

第1组锁与重复（Lock and repeat）

锁（LOCK）编码为：F0H。用于互斥访问共享内存的操作。

非零时重复（REPNE/REPNZ）编码为：F2H。用于字符串操作指令。

为零时重复（REP/REPE/REPZ）编码为：F3H。用于字符串操作指令。

第2组段覆盖与分支提示

段覆盖（Segment override）：CS、SS、DS、ES、FS、GS的段覆盖前缀的编码分别是2EH、36H、3EH、26H、64H、65H（在分支指令中使用未定义）。

分支提示（Branch hints），只用于条件分支指令Jcc。提示分支不发生编码为2EH；提示分支发生编码为3EH。

第3组操作数长度覆盖（Operand-size override）编码为66H。用于在16位与32位操作数切换。

第4组地址长度覆盖（Address-size override）编码为67H。用于在16位与32位地址切换。

指令码

长度为1、2或3字节，此外ModR/M中还可能有3位。对于双字节指令码或三字节指令码，其中的第1个字节为0FH，用于与指令前缀区分。

ModR/M与SIB

许多指令的内存操作数需要使用ModR/M字节作为寻址模式说明符。其中的mod与r/m组合，共有32个值，表示8个寄存器与24种寻址模式。reg/opcode表示寄存器号或者额外的3位指令码，其具体含义依赖基本指令码。Mod与R/M的5位表示的第一操作数（源与目的操作数中寻址方式更复杂的那个操作数，指令码中的“方向位”direction bit(d)给出源或目的操作数哪个是第一操作数）的寻址方式如下：

寻址方式 Mod [展开]R/M

某些ModR/M字节表示的寻找模式，需要SIB字节来补充寻址方式。scale表示比例系数；index表示变址寄存器号；base表示基址寄存器号。使用scale与index的5位定义比例变址寄存器如下：

比例变址 Scale [展开]Index

3位base表示的基址寄存器号，定义如下：

EAX ECX EDX EBX ESP [\*] ESI [展开]EDI

在汇编程序设计中，一般把第1操作数的寻址方式总结为如下8种：

寻址方式 英文术语 [展开]举例

综合指令格式中的ModR/M与SIB两个字节的语义规定，指令的第1操作数的寻址方式可总结为4种物理实现：

立即数：表示在指令的“立即数”部分。包括了直接寻址，即立即数作为内存的地址。

寄存器操作数：Mod为11B，根据R/B部分的值、指令码、操作数长度属性，确定具体的寄存器号。

基址相对寻址：即[Reg+disp8或disp32]。包括了寄存器间接寻址。这种情况计算第1操作数地址时使用了1个寄存器。

基址加比例变址的相对寻址：即[BaseReg+IndexReg\*scale+disp8或disp32]。这种情况计算第1操作数地址时使用了2个寄存器。

位移与立即数

某些寻址方式需要给出位移值。有些指令需要给出立即数作为操作数。

Intel64和IA32架构的指令编码格式是图2-1格式的子集。指令有这几个部分组成：可选的指令前缀（顺序不限）、主操作码字节（最多3个字节）以及寻址方式说明（addressing-form specifier）字节（如果需要的话）。寻址方式说明由ModR/M字节、SIB字节、位移量（如果需要的话）和立即数（如果需要的话）组成。

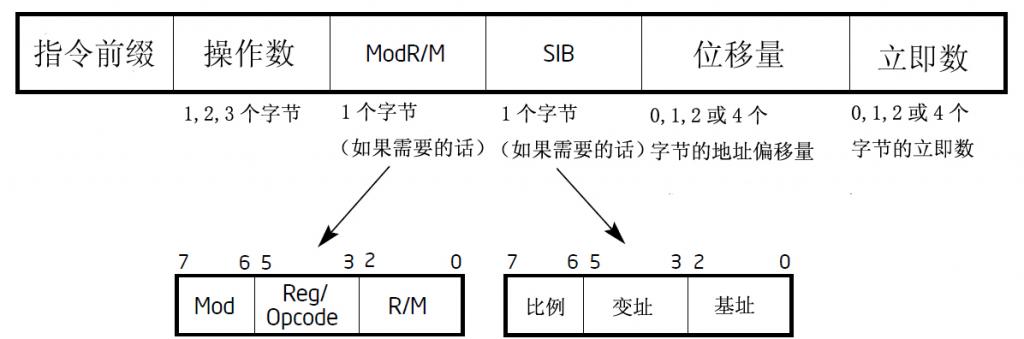


图2-1Intel64和IA-32架构指令格式

2.1.1 指令前缀

指令前缀分成四块，每块都有一组前缀代码。对于每个指令来说，四块中每块只能使用一个前缀。这四块的前缀的顺序是不限定的。

•第一块

—锁前缀和重复前缀：

•LOCK锁前缀代码为F0H

•REPNE/REPNZ前缀代码为F2H。该前缀仅用于字符串和输入/输出指令(F2H在某些指令中还被用作强制前缀(mandatory prefix)).

• REP或者REPE/REPZ前缀代码为F3H。该前缀仅用于字符串和输入/输出指令(F3H在某些指令中还被用作强制前缀).

•第二块

—段超越前缀

• 2EH——CS段超越（在分支指令中使用未定义）

• 36H——SS段超越（在分支指令中使用未定义）

• 3EH——DS段超越（在分支指令中使用未定义）

• 26H——ES段超越（在分支指令中使用未定义）

• 64H——FS段超越（在分支指令中使用未定义）

• 65H——GS段超越（在分支指令中使用未定义）

—分支预测（branch hints）前缀

• 2EH——分支被使用(只用于条件转移JCC指令)

• 3EH——分支不被使用(只用于条件转移JCC指令)

• 第三块

•调整操作数大小(operand-size override)前缀代码为66H(在一些指令中66H被用作强制前缀)

• 第四块

•67H——调整地址大小(address-size override)前缀

锁前缀（F0H）在多处理器情况下会迫使一个操作必须互斥地使用共享内存。（详见《“Instruction Set Reference, A-M,”》 第三章“LOCK—Assert LOCK# Signal Prefix”）

重复前缀（F2H,F3H）会使指令对字符串中的每一个元素都重复执行。该前缀只在字符串和I/O指令(MOVS,CMPS, SCAS,LODS,STOS,INS,OUTS)中使用。在其他未定义的操作码前使用重复前缀没有定义，这种使用会导致不可预见的后果。

一些指令会将F2H/F3H作为强制前缀来表示不同的功能（express distinct functionality）。强制前缀通常应置于其他可选前缀后面（例外情况在2.2.1“REX Prefixes”中讨论）。

分支预测前缀（2EH,3EH）允许程序预测一个分支最有可能的路径。此前缀只在条件转移指令（JCC）中使用。在其他未定义的操作码前使用分支预测前缀没有定义，这种使用会导致不可预见的后果。

调整操作数大小前缀（66H）允许程序在16位和32位操作数之间切换操作数。这两种大小都可以作为缺省值，使用本前缀可以切换为非缺省的那个大小值。

一些SSE2/SSE3/SSSE3/SSE4指令和主操作码为3字节的指令可能会将66H作为强制前缀来表示不同的功能。强制前缀通常应置于其他可选前缀后面（例外情况在2.2.1“REX Prefixes”中讨论）。

66H前缀的其他使用保留，如果使用可能会导致不可预见的后果。

调整地址大小前缀（67H）允许程序在16位和32位地址之间切换。这两种大小都可以作为缺省值，使用本前缀可以切换为非缺省的那个大小值。当指令的操作数是内存中不存在时，使用该前缀和/或者未定义的操作码是未定义的，这种使用会导致不可预见的后果。

.

2.1.2 操作码

主操作码可以是1个,2个或3个字节长。有时会有另外的3位操作码存在于ModR/M字节中。主操作码中还可以为定义更小的段(fields)，比如操作方向、位移量（displacements）大小、寄存器编码、条件码(condition codes)或者符号扩展(sign extension)。操作码中的段随操作的类型不同而有很大的不同。

通用指令和SIMD指令的2个字节的操作码格式组成如下：

•一个逃逸码(escape opcode)0FH作为主操作码和一个第二操作码字节。或者

•一个强制前缀（66H、F2H或F3H），一个逃逸码字节和一个第二操作码字节。

举例说明,CVTDQ2PD的操作码序列为F3 0F E6。第一个字节就是强制前缀（不是重复前缀）。

通用指令和SIMD指令的3个字节的操作码格式组成如下：

•一个逃逸码0FH作为主操作码，加上两个额外的操作码字节。或者

•一个强制前缀（66H、F2H或F3H），一个逃逸码字节，加上两个额外的操作码字节。

举例说明，XMM寄存器的PHADDW指令的操作码序列为66 0F 38 01。第一个字节就是强制前缀。

所有有效的操作码列于附录A和附录B.

2.1.3 ModR/M和SIB字节

很多引用内存操作数的指令需要一个说明寻址方式的字节（称为ModR/M）紧跟在主操作码后面。ModR/M字节包含下面三个信息段（译注：见图2-1）：

•Mod段和r/m段形成32种可能值：8种寄存器和24种寻址方式。

•Reg/Opcode段或者指定一个寄存器或者是主操作码的3位附加信息。Reg/opcode段的具体作用由主操作码决定。

•r/m段可以指定一个寄存器作为操作数，或者和mod段组合起来指定一种编码方式。有时候，这种mod段和r/m段的组合可以被用来表示一些指令的操作码信息。

一些ModR/M字节还需要第二个寻址字节（SIB）。32位的基址变址（base-plus-index）和比例变址(scale-plus-index)寻址方式需要一个SIB字节。SIB字节包含下面段（译注：见图2-1）：

•比例段指出比例倍数(scale factor)

•变址段指出变址寄存器的寄存器数字

•基址段指出基址寄存器的寄存器数字

参看2.1.5节ModR/M和SIB字节的编码

2.1.4位移量和立即数字节

一些寻址方式在ModR/M(或者SIB，如果有的话)后面会有一个位移量。如果有位移量的话，它可以是1、2或4字节。如果一个指令指定一个立即数，那么这个立即数会跟在位移量字节后面，立即数可以是1、2和4字节。

2.1.5 ModR/M和SIB字节组成的寻址方式

由ModR/M和SIB字节组成的值和其对应的寻址方式列于表2-1到表2-3：由ModR/M字节指定的16位寻址方式列于表2-1；32位寻址方式列于表2-2；表2-3列出了由SIB字节指定的32位寻址方式。假如ModR/M字节中reg/opcode段表示一个扩展操作码，有效编码见附录B.

在表2-1和表2-2中，有效地址（Effective Address）列中列出了由Mod和R/M组成的可以赋给指令的第一个操作数的32种有效地址。前面24种指出了一个内存的位置；后面的8种（Mod=11B）是指出了通用寄存器、MMX寄存器和XMM寄存器。

表2-1和表2-2中的Mod和R/M列给出了要获得第一列这种有效地址方式的二进制编码。比如，看Mod=11B，R/M=000B那一行，该行标明的通用寄存器为EAX,AX和AL;MMX寄存器为MM0;XMM寄存器为XMM0.具体到底使用哪个寄存器是由操作码字节和操作数大小属性决定。

我们再来看表2-1或2-2任一个表的第7行（标有“REG=”的那行）。该行指出如何使用3位的Reg/Opcode段来获得第二个操作数的位置。第二个操作数必须是通用寄存器、MMX寄存器或者XMM寄存器。第1到5行列出了表中值所对应的寄存器。和上面一样，使用哪种寄存器是由操作码字节和操作数大小属性决定的。

如果指令不需要第二个操作数，Reg/Opcode段可以被用作操作码的扩展。由表的第6行（标有“/DIGIT(OPCODE)”的那行）标出。注意该行值是十进制形式的。

表2-1和表2-2的主体部分（在标有“Value of ModR/M Byte (in Hexadecimal)”的下面）包含一个由32\*8数组所表示的256个ModR/M值（16进制）。第3/4/5位由字节所在列指出，第0/1/2位和第6/7位由行指出。下图演示如何计算出表中值的。

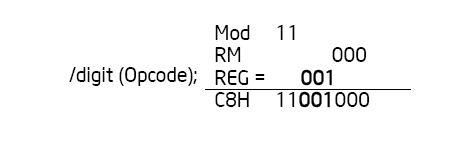
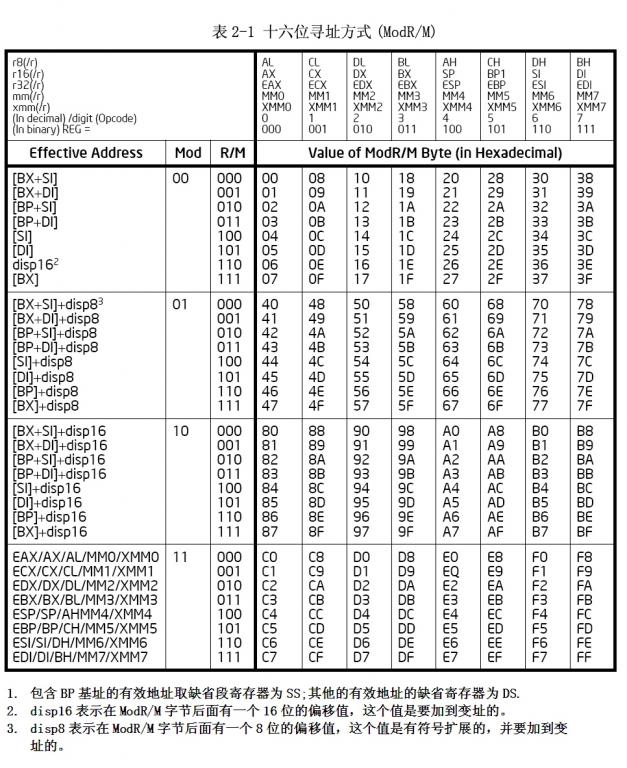


图2-2 解释如何得到ModR/M(C8H)字节



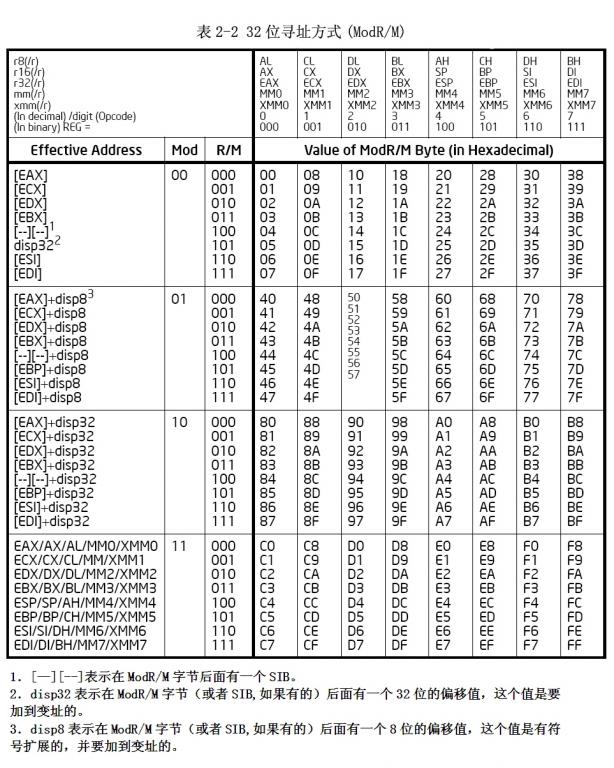
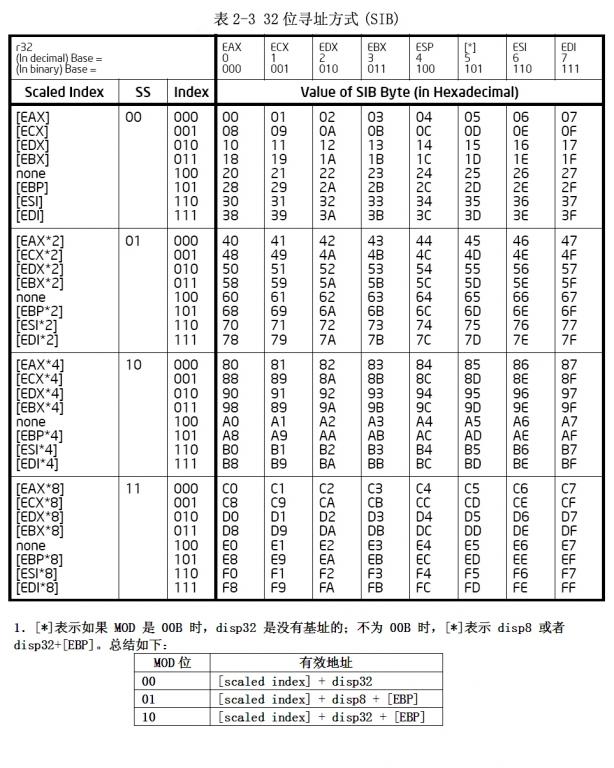


表2-3显示的是256中SIB字节的可能值（16进制）。用作基址的通用寄存器列在表的上部，并同时显示了在SIB字节中的对应值。表的主体部分的行是作为变址的寄存器和比例倍数（由SIB字节的第6/7位表示）.



学习汇编语言时，我们都是从寻址方式开始的。所谓寻址方式就是指令中操作数是如何指定的。在16位时，寻址方式有：

1.立即数寻址

操作数直接存放在指令中，紧跟在操作码之后。

2.寄存器寻址

操作数在寄存器中，指令指定寄存器号。

3.存储器寻址，存储器寻址方式是说操作数存放在内存之中，指定这些内存的有效地址（effective address）有下面这几种方法。

3.1 直接寻址

直接寻址时，操作数的有效地址紧跟操作码之后。比如：MOV AX, [0X1000]

3.2 寄存器间接寻址

操作数的有效地址存放在基址寄存器BX和BP、或者变址寄存器SI和DI中。

EA = 16 \* DS + SI

或EA = 16 \* DS + DI

或EA = 16 \* DS + BX

或EA= 16 \* SS + BP (注意：BP的缺省段寄存器为SS，其他都为DS)

3.3 寄存器相对寻址

操作数的有效地址是一个基址或变址寄存器的内容和指令中指定的8位或16为偏移量的和。注意这个偏移量是带符号的，即可正可负。

EA = 16 \* DS + SI + DISP8/16

或 EA = 16 \* DS + DI + DISP8/16

或 EA = 16 \* DS + BX + DISP8/16

或 EA = 16 \* SS + BP + DISP8/16

3.4 基址变址寻址

操作数的有效地址是一个基址寄存器和一个变址寄存器的内容之和。

EA = 16 \* DS + BX + SI或 DI

或 EA = 16 \* SS + BP + SI或 DI

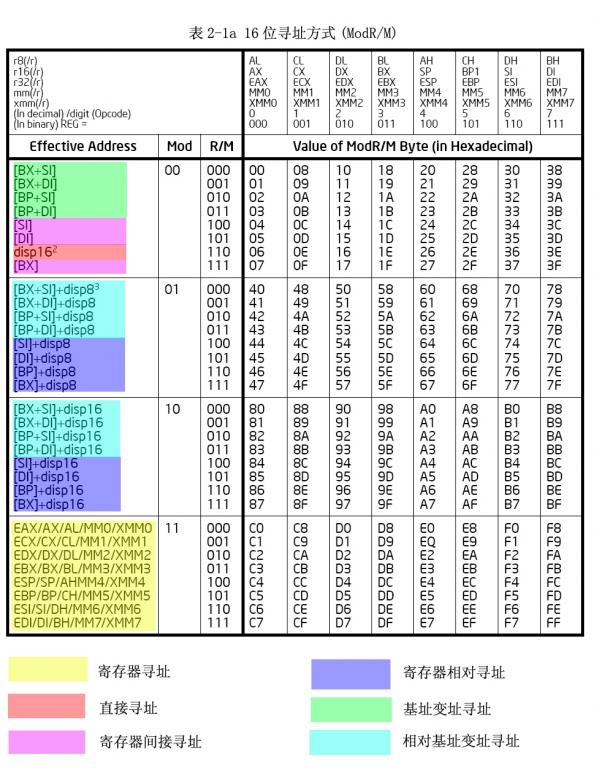
3.5 相对基址变址寻址

操作数的有效地址是一个基址寄存器和一个变址寄存器的内容和8位或16位偏移量之和。同上面一样，这个偏移量是带符号的，即可正可负。

EA = 16 \* DS + BX + SI(或 DI) + DISP8/16

或 EA = 16 \* SS + BP + SI(或 DI) + DISP8/16

上面的这些寻址方式，我将它在表2-1中表示了出来，见表2-1a。当初学习寻址方式时总是一头雾水，现在将它和I32的指令码格式联系在一起总算明白了。:)



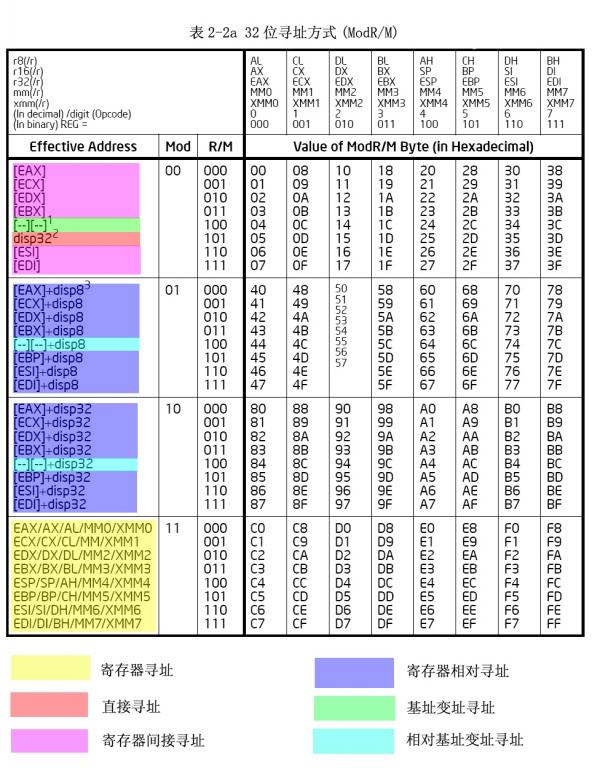
上图中我们还发现一个问题：寄存器间接寻址中可以使用SI、DI、BX、BP这四个寄存器。但是我们看到Mod=00时只能表示SI、DI、BX三个寄存器，那么BP是如何表示的呢？原来它是通过在MOD=01和10时，设置disp8和disp16偏移值来实现的。所以在编译下面两条指令时可能差别很大的。

指令 编码

Mov ax, [bx] 8B 07

Mov ax, [bp] 8B 46 00

32位寻址方式和16位上述的寻址方式基本一致见表2-2a。



和16位寻址方式的几个区别：

a.32位的寄存器间接寻址可以使用除ESP外的所有通用寄存器，而不是16位时的四个寄存器（SI、DI、BX、BP）。

b.32位的寄存器相对寻址、基址变址寻址、相对基址变址寻址都可以使用除ESP外的所有的通用寄存器。

c.由于32位寻址方式添加了一位SIB字节（见表2-3），所以在基址变址寻址和相对基址变址方式时，可以使用一个新的特性：比例。一共有1、2、4、8四种比例方式。

参考文献：