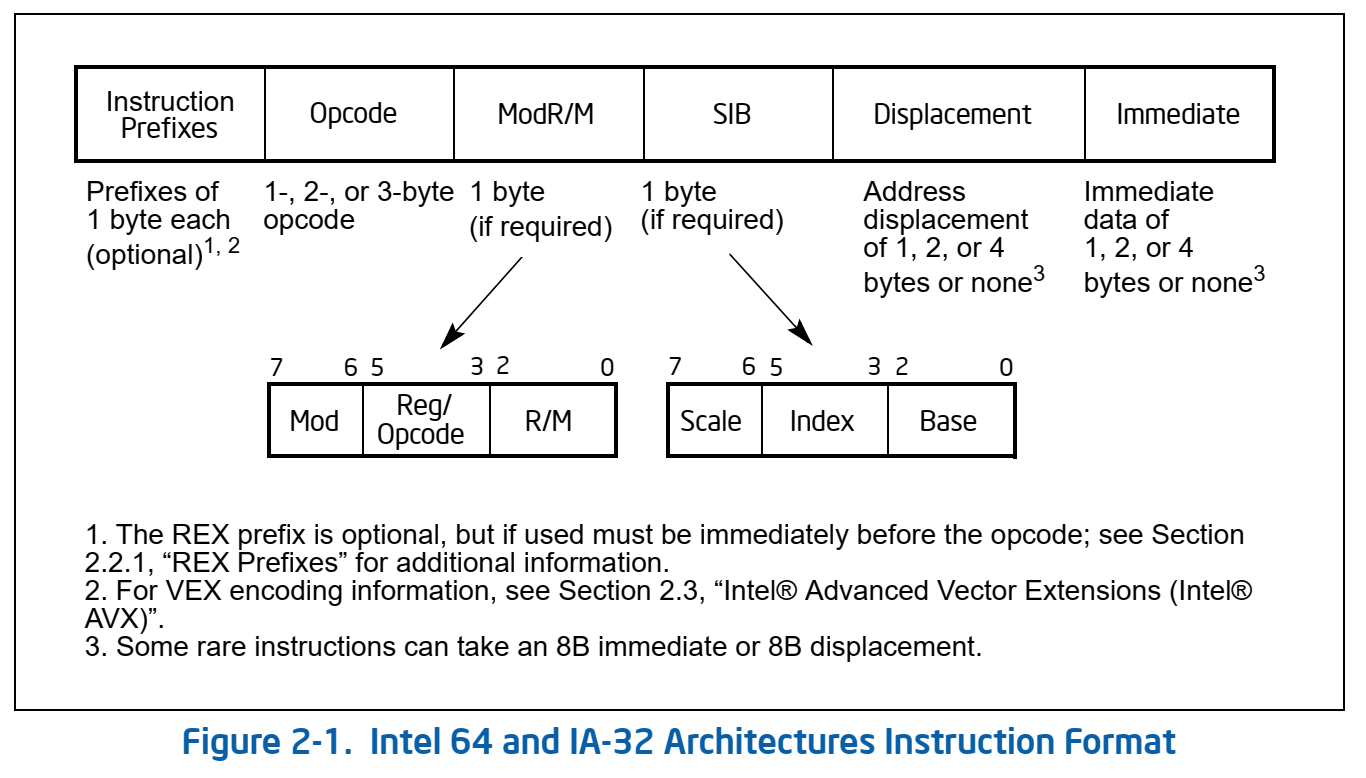
**Intel汇编指令格式解析**

来源：[Intel汇编指令格式解析 - dchao - 博客园 (cnblogs.com)](https://www.cnblogs.com/dongc/p/10727457.html)

**环境：**

　　win7\_x64旗舰版、VS2015企业版

**一、Intel保护模式、实地址模式和虚拟8086模式指令格式(x86)**



**图在Intel手册Volume2 2.1节**

　　1.1）Instruction Prefixes：指令前缀，可选项，每个前缀一个字节，可选0个前缀到4个不等；指令前缀分为四组，每组都允许设置指定的前缀代码。

**Group 1： 有两种：**

— Lock and repeat prefixes: 锁定和重复前缀。

• LOCK prefix is encoded using F0H.

• REPNE/REPNZ prefix is encoded using F2H. Repeat-Not-Zero prefix applies only to string and

input/output instructions. (F2H is also used as a mandatory prefix for some instructions.)

• REP or REPE/REPZ is encoded using F3H. The repeat prefix applies only to string and input/output instructions. F3H is also used as a mandatory prefix for POPCNT, LZCNT and ADOX instructions.

— BND prefix is encoded using F2H if the following conditions are true:

• CPUID.(EAX=07H, ECX=0):EBX.MPX[bit 14] is set.

• BNDCFGU.EN and/or IA32\_BNDCFGS.EN is set.

• When the F2 prefix precedes a near CALL, a near RET, a near JMP, a short Jcc, or a near Jcc instruction.

**Group 2：**

—段覆盖前缀Segment override prefixes:

• 2EH—CS segment override (use with any branch instruction is reserved).

• 36H—SS segment override prefix (use with any branch instruction is reserved).

• 3EH—DS segment override prefix (use with any branch instruction is reserved).

• 26H—ES segment override prefix (use with any branch instruction is reserved).

• 64H—FS segment override prefix (use with any branch instruction is reserved).

• 65H—GS segment override prefix (use with any branch instruction is reserved)

—分支选择提示前缀Branch hints:

• 2EH—Branch not taken (used only with Jcc instructions).

• 3EH—Branch taken (used only with Jcc instructions).

**Group 3：66H，操作数大小覆盖前缀。**

**Group 4：67H，地址大小覆盖前缀。**

　　1.2）Opcode：操作码，这是唯一不可省略的项，1、2或3个字节，在某些情况下会有ModR/M中的额外3bit作为补充，这三个位是Reg/Opcode域。

1.3）ModR/M

许多引用内存中操作数的指令都有一个寻址形式说明符字节（称为ModR/M字节）在主操作码之后。ModR/M字节包含三个信息字段： Mod，Reg/Opcode， R/M。

• 字段Mod有2bit，与R/M字段的3位组合，可以产生32个可能的数值：可以表示8个寄存器和24种寻址模式

• 字段 Reg/Opcode有3bit，指定一个寄存器(编号)或作为opcode编码的额外3位。该字段的用途由主操作码Opcode指定。

• 字段 R/M有3bit，可以指定一个寄存器作为操作数，或者或mod字段联合编码，指定寻址模式。有时，mod字段和r/m字段的某些组合用于表示某些指令的操作码opcode信息。

　　　　ModR/M字节的某些编码需要第二个寻址字节（SIB字节），32位寻址的base-plus-index和scale-plus-index形式需要SIB字节。SIB字节包括以下字段：

* "scale"字段指定比例因子。
* "index"字段指定索引寄存器的寄存器索引
* "base"字段指定基本寄存器的寄存器索引
* 有关ModR/M和SIB字节的编码，请参见第2.1.5节

　MOV **Opcode**：

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Opcode** | **Instruction** | **Op/En** | **64-Bit Mode** | **Compat/Leg Mode** | **Description** |
| 88 /*r* | MOV *r/m8,r8* | MR | Valid | Valid | Move *r8* to *r/m8.* |
| REX + 88 /*r* | MOV *r/m8*\*\*\*,*r8*\*\*\* | MR | Valid | N.E. | Move *r8* to *r/m8.* |
| 89 /*r* | MOV *r/m16,r16* | MR | Valid | Valid | Move *r16* to *r/m16.* |
| 89 /*r* | MOV *r/m32,r32* | MR | Valid | Valid | Move *r32* to *r/m32.* |
| REX.W + 89 /*r* | MOV *r/m64,r64* | MR | Valid | N.E. | Move *r64* to *r/m64.* |
| 8A /*r* | MOV *r8,r/m8* | RM | Valid | Valid | Move *r/m8* to *r8.* |
| REX + 8A /*r* | MOV *r8\*\*\*,r/m8\*\*\** | RM | Valid | N.E. | Move *r/m8* to *r8.* |
| 8B /*r* | MOV *r16,r/m16* | RM | Valid | Valid | Move *r/m16* to *r16.* |
| 8B /*r* | MOV *r32,r/m32* | RM | Valid | Valid | Move *r/m32* to *r32.* |
| … | … | … | … | … | … |

　MOV 操作数编码(Op/En)：

**Instruction Operand Encoding**[**¶**](https://www.felixcloutier.com/x86/mov#instruction-operand-encoding)

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Op/En** | **Operand 1** | **Operand 2** | **Operand 3** | **Operand 4** |
| MR | ModRM:r/m (w) | ModRM:reg (r) | NA | NA |
| RM | ModRM:reg (w) | ModRM:r/m (r) | NA | NA |
| FD | AL/AX/EAX/RAX | Moffs | NA | NA |
| TD | Moffs (w) | AL/AX/EAX/RAX | NA | NA |
| OI | opcode + rd (w) | imm8/16/32/64 | NA | NA |
| MI | ModRM:r/m (w) | imm8/16/32/64 | NA | NA |

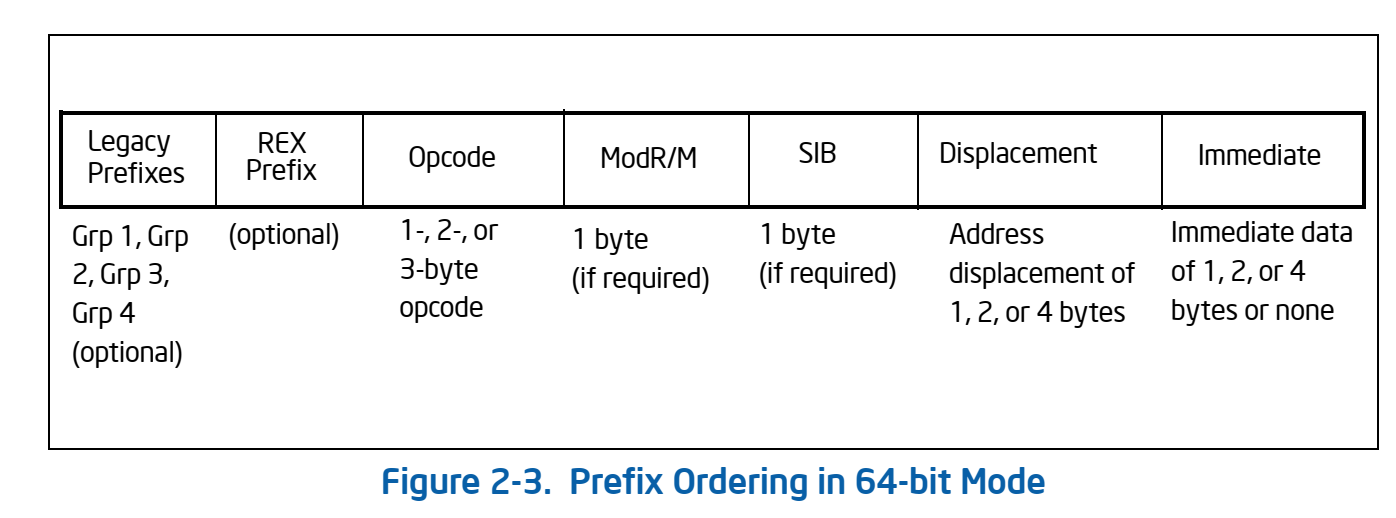
　　　　　对于89 /r，Op/En为MR，Reg/Opcode表示第二个操作数Operand2寄存器。

　　　　而对于8B /r，Op/En为RM，Reg/Opcode则表示第一个操作数Operand1寄存器。

　　1.4）SIB：定义ModR/M的寻址方式的补充寻址方式，使用"Base + Scaled Index"格式。

　　1.5）Displacement：偏移，可选，0，1，2，4个字节。

　　1.6）Immediate：立即数，可选，0，1，2，4个字节。

**二、IA-32E模式**

**图在Intel手册Volume2 2.2.1章节 P42**

　　2.1）IA-32e模式有两个子模式

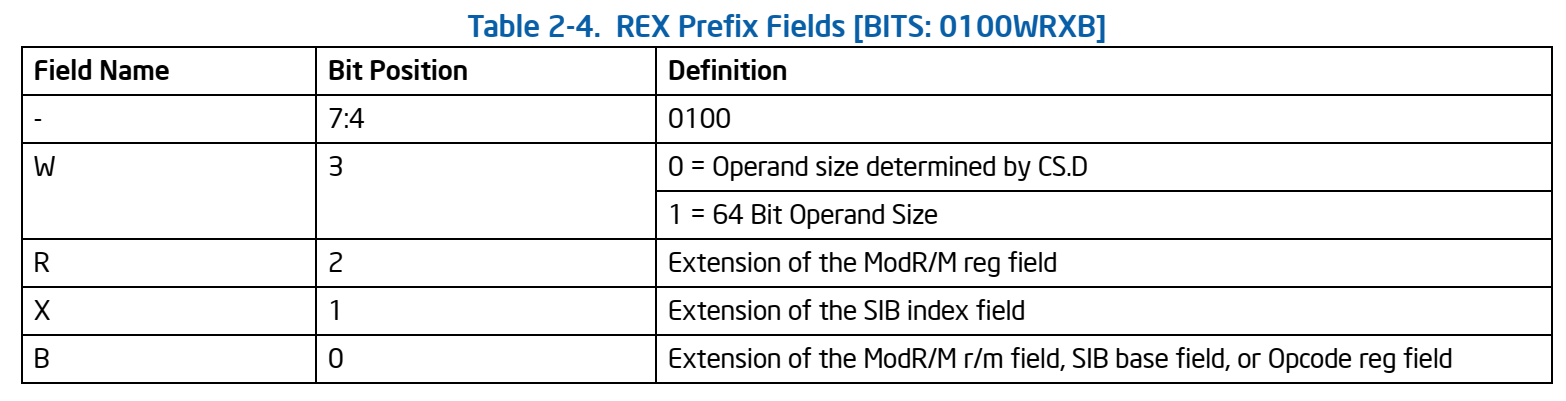
　　　　Compatibility Mode：使64位操作系统能够不加修改地运行大多数传统保护模式软件。

　　　　64-Bit Mode：使64位操作系统能够运行为访问64位地址空间而编写的应用程序。

　　2.2）REX Preflx

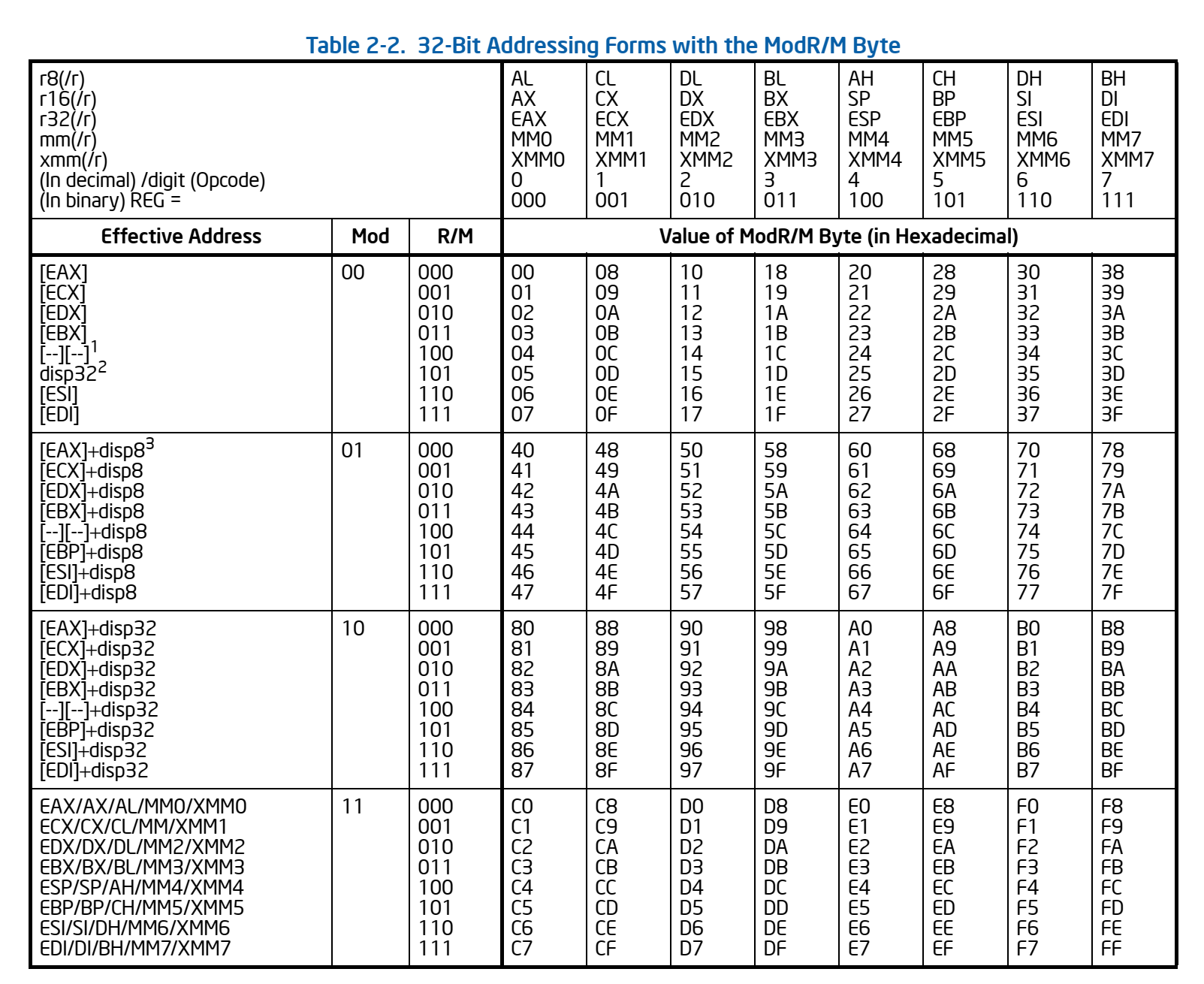
  REX前缀是在64位模式下使用的指令前缀字节，与x86模式的不同之处就是多了这个REX Preflx，其他基本与x86相同。它有以下作用：

* 指定GPRs和SSE寄存器
* 指定64位操作数大小
* 指定扩展寄存器



　　　　并非所有指令都需要64位模式下的REX前缀。 仅当一条指令引用扩展寄存器之一或使用64位操作数时，才需要前缀。 如果在没有含义的情况下使用REX前缀，则将其忽略。

**三、使用ModR/M字节的32位寻址**



**图在Intel手册Volume2 2.5.1章节Addressing-Mode Encoding of ModR/M and SIB Bytes**

　　解释：

　　3.1）[--][--]：表示使用SIB结构。

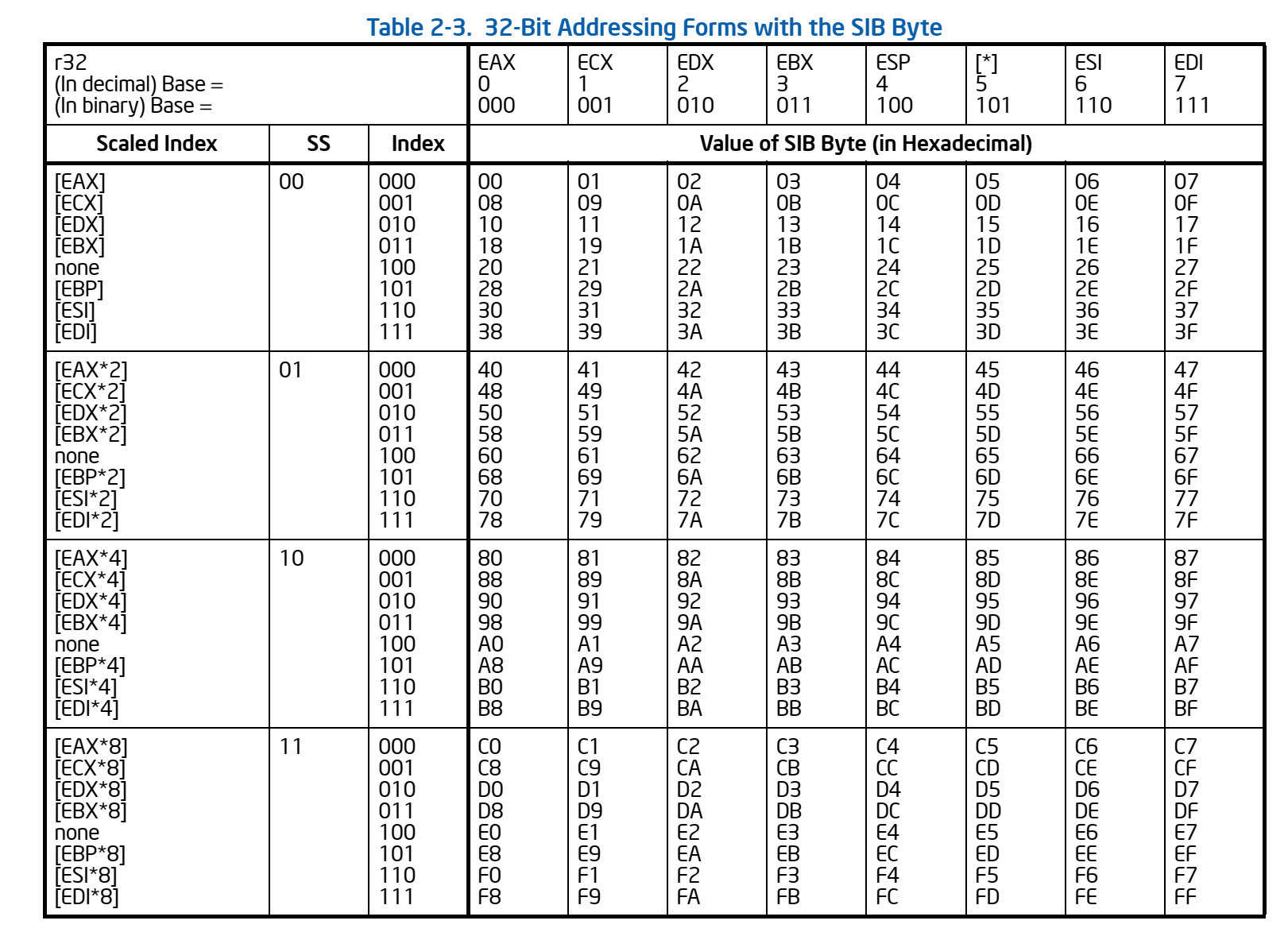
　　3.2）disp32：表示32位偏移。

　　3.3）[--][--]+disp8：表示使用SIB结构，且SIB结构后面有一个8位的偏移。

　　3.4）[--][--]+disp32：表示使用SIB结构，且SIB结构后面有一个32位的偏移。

|  |  |  |
| --- | --- | --- |
| ModR/M.mod | 寻址模式 | 描述 |
| 00 | [base] | 提供 [base] 形式的 memory 寻址 |
| 01 | [base + disp8] | 提供 [base + disp8] 形式的 memory 寻址 |
| 10 | [base + disp32] | 提供 [base + disp32] 形式的 memory 寻址 |
| 11 | register | 提供 register 寻址。 |

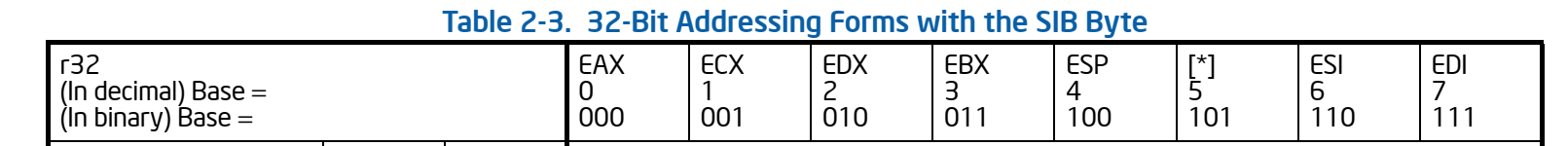
**四、使用SIB字节的32位寻址**



**图在Intel手册Volume2 2.5.1章节Addressing-Mode Encoding of ModR/M and SIB Bytes**

　　SIB结构使用"Base + Scaled Index"格式。

Base的可选值为：



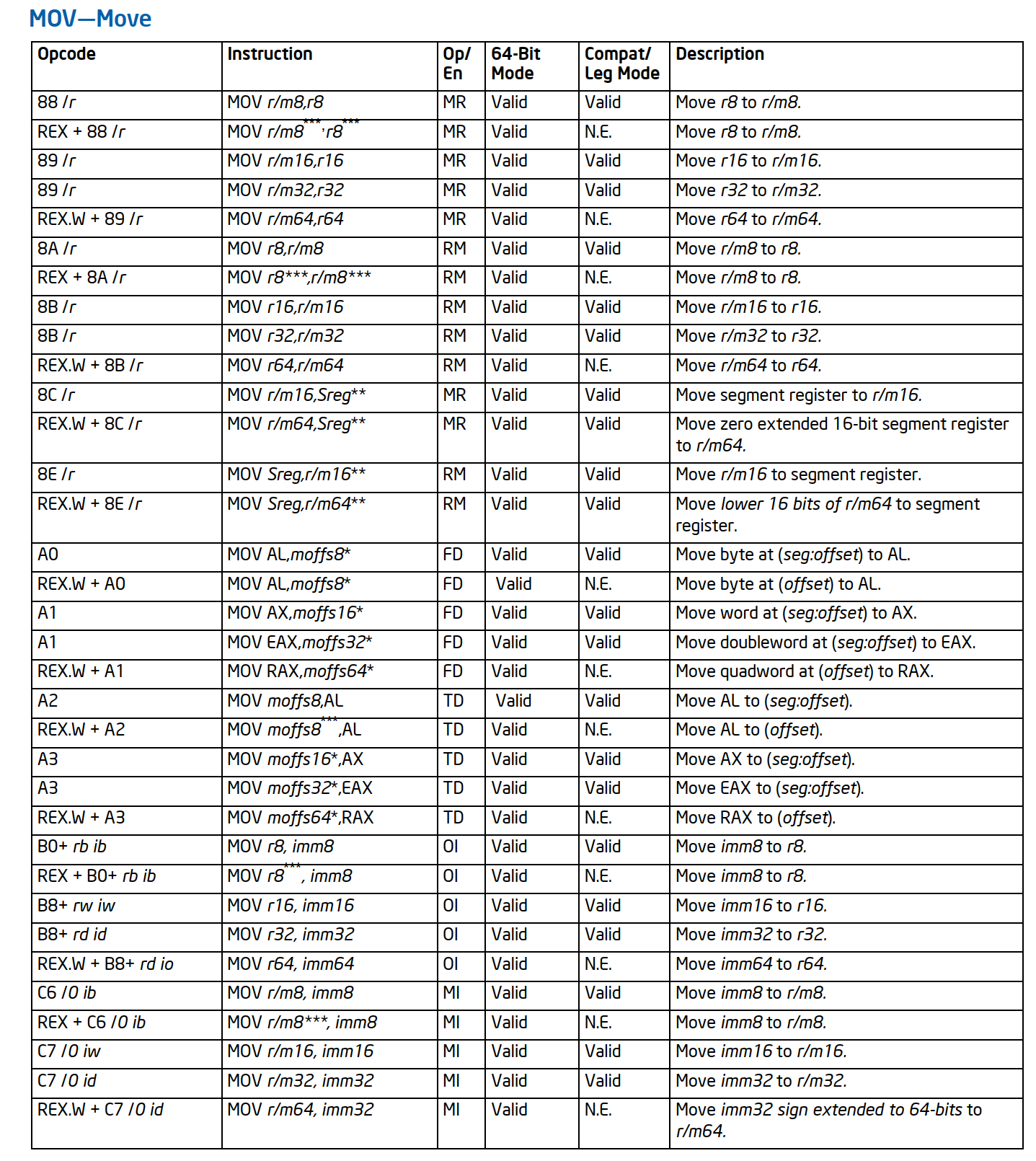
　　Scaled Index的可选值为：none、[REG]、[REG\*2]、[REG\*4]、[REG\*8]，REG为实际的某个寄存器。

　　none表示没有Scaled Index，即Scaled Index的为0。

　　[REG]表示使用一个寄存器如[EAX]、[ECX]和[EDX]等等。

　　[REG\*2]表示使用寄存器乘以2如[EAX\*2]、[ECX\*2]和[EDX\*2]等等。

**五、MOVE-Move指令格式**



**图在Intel手册Volume2 4.3章节MOV-Move小节**

　　解释：

　　5.1）r：寄存器，r8表示8位寄存器，r16表示16位寄存器，r32和r64依次类推。

　　5.2）m：内存地址，m8表示8位地址，m16表示16位地址，m32和m64依次类推。

　　5.3）r/m：寄存器或内存

　　5.4）/r：表示操作码有ModR/M结构，且ModR/M结构的Reg/Opcode域为Reg，表示第二个操作码寄存器。

　　5.4）REX：表示该操作码有REX Preflx字段。

　　5.5）REG.W：表示该操作码有REX Preflx字段且W位值为1。

　　5.6）rb：表示使用byte寄存器，即8位寄存器。

　　5.7）ib：表示使用byte立即数，即8位立即数。

　　5.8）rw：表示使用word寄存器，即16位寄存器。

　　5.9）iw：表示使用word立即数，即16位立即数。

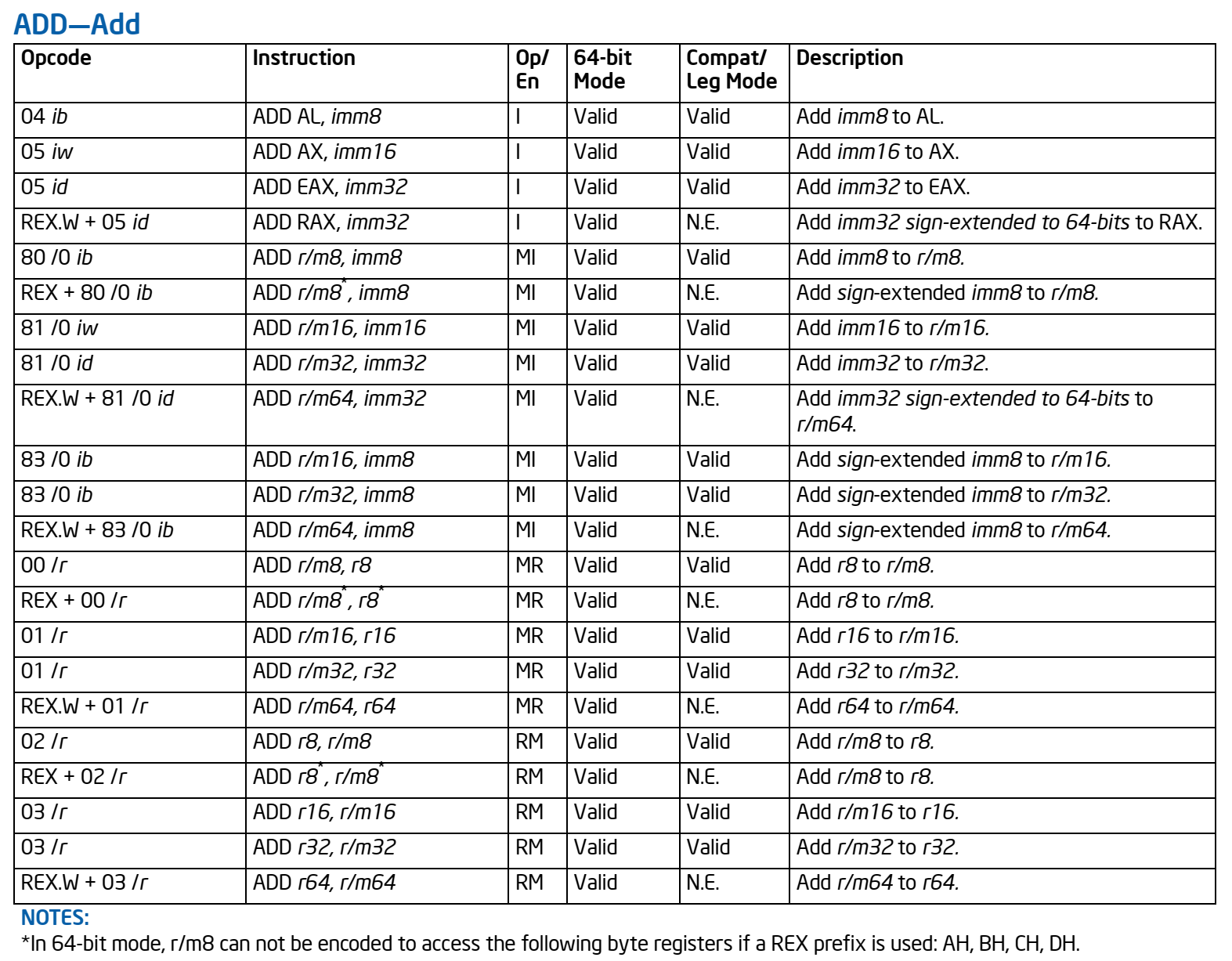
　　5.10）rd：表示使用dword寄存器，即32位寄存器。

　　5.11）id：表示使用dword立即数，即32位立即数。

　　5.12）/数字：数字在0-7之前，例如/0或/7，表示指定有ModR/M选项，且只使用r/m，reg字段包含提供指令操作码扩展名的数字，例如/4表示reg字段为100B。

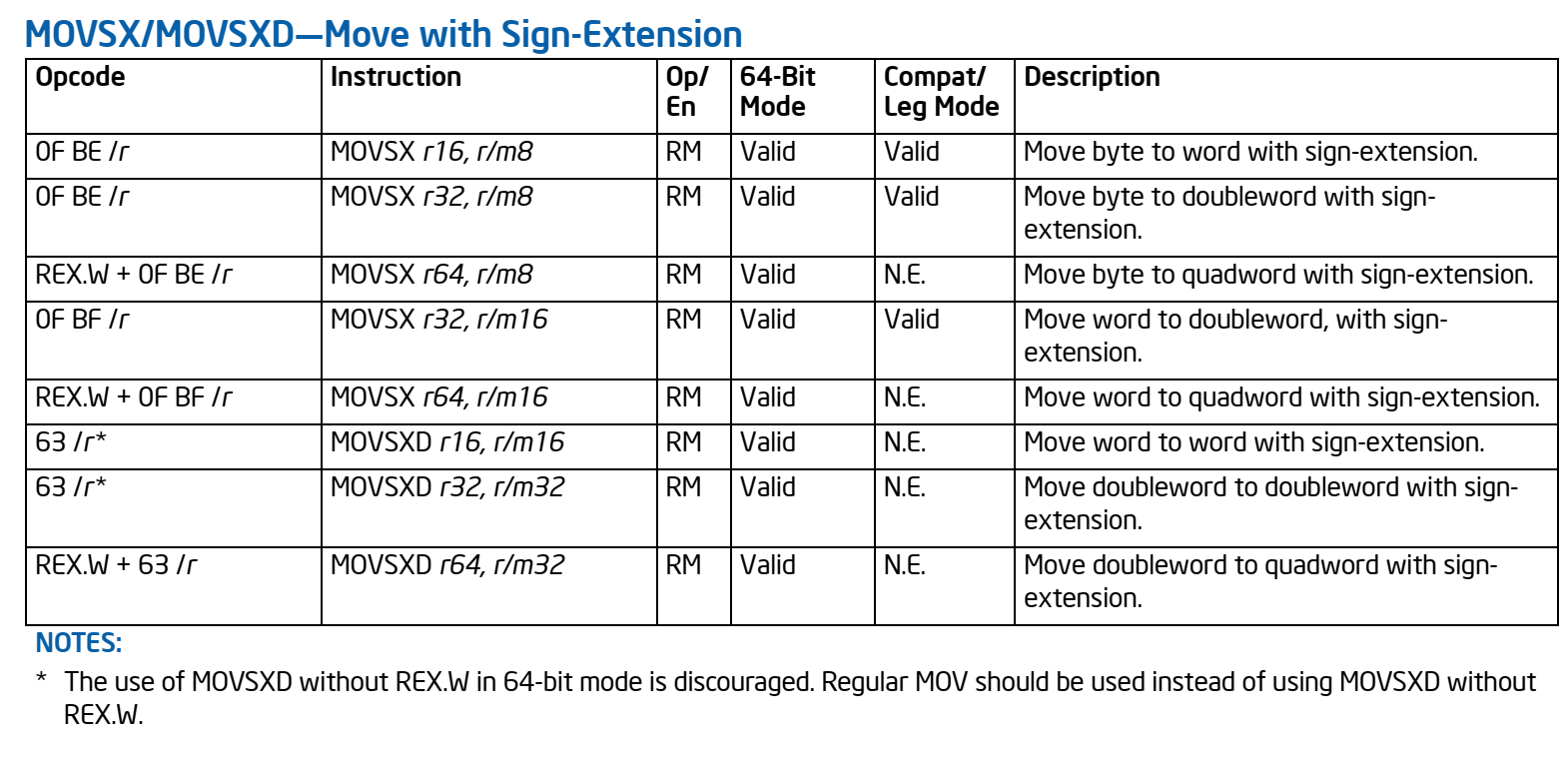
**完整解释请看Intel手册Volume2第3.1.1.1章节Opcode Column in the Instruction Summary Table (Instructions without VEX Prefix)**

**六、ADD-Add指令格式**



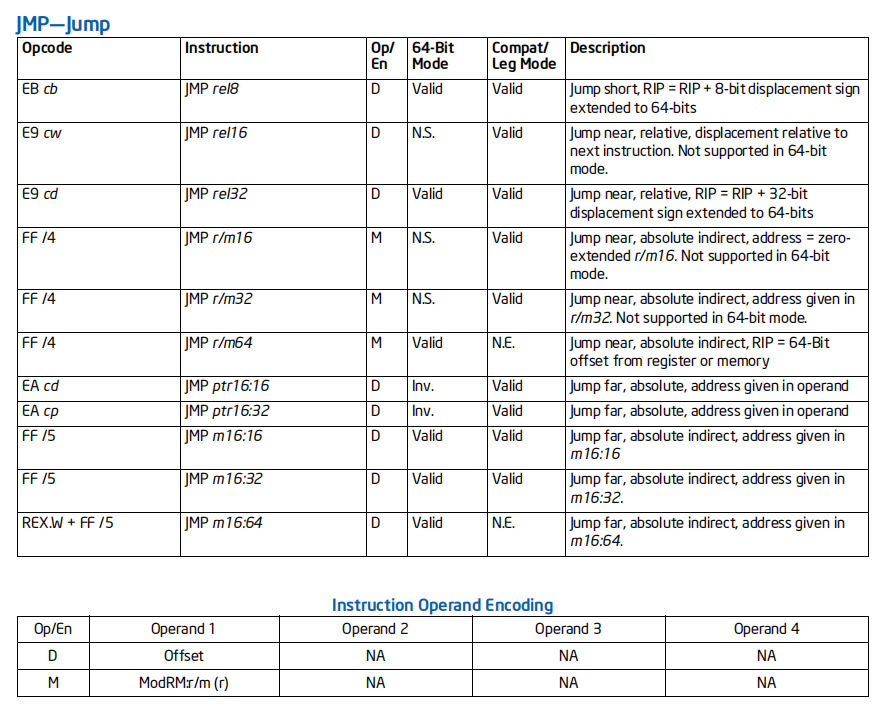
**图在Intel手册Volume2 3.2章节ADD-Add小节**

**七、MOVSX/MOVSXD指令格式**



**图在Intel手册Volume2 4.3章节MOVSX/MOVSXD—Move with Sign-Extension小节**

**八、JMP—Jump指令格式**

****

解释：

|  |  |  |
| --- | --- | --- |
| 代码 | 指令 | 说明 |
| EB cb | JMP rel8 | 相对短跳转，位移量相对于下一条指令，跳转地址为RIP + 2 + rel8 |
| E9 cw | JMP rel16 | 相对近跳转，位移量相对于下一条指令，跳转地址为RIP + 3 +rel16 |
| E9 cd | JMP rel32 | 相对近跳转，位移量相对于下一条指令，跳转地址为RIP + 5 + rel32 |
| FF /4 | JMP r/m16 | 绝对间接近跳转，地址由 r/m16 给出 |
| FF /4 | JMP r/m32 | 绝对间接近跳转，地址由 r/m32 给出 |
| FF /4 | JMP r/m64 | 绝对间接近跳转，地址由 r/m64 给出，跳转地址为\*(REG + r/m64)，/4表示Reg/Opcode为100B |
| EA cd | JMP ptr16:16 | 绝对远跳转，地址由操作数给出 |
| EA cp | JMP ptr16:32 | 绝对远跳转，地址由操作数给出 |
| FF /5 | JMP m16:16 | 绝对间接远跳转，地址由 m16:16 给出 |
| FF /5 | JMP m16:32 | 绝对间接远跳转，地址由 m16:32 给出 |

**8.1）示例（x64环境）**

**0x0000000000405f10 ff 25 32 52 20 00    jmpq   \*0x205232(%rip)**

属于FF /4 JMP r/m64，跳转地址为=\*(rip + 0x205232)，其中ModR/M=0x25=00100101B，Mod=00B，Reg/Opcode=/4=100B，R/M=101

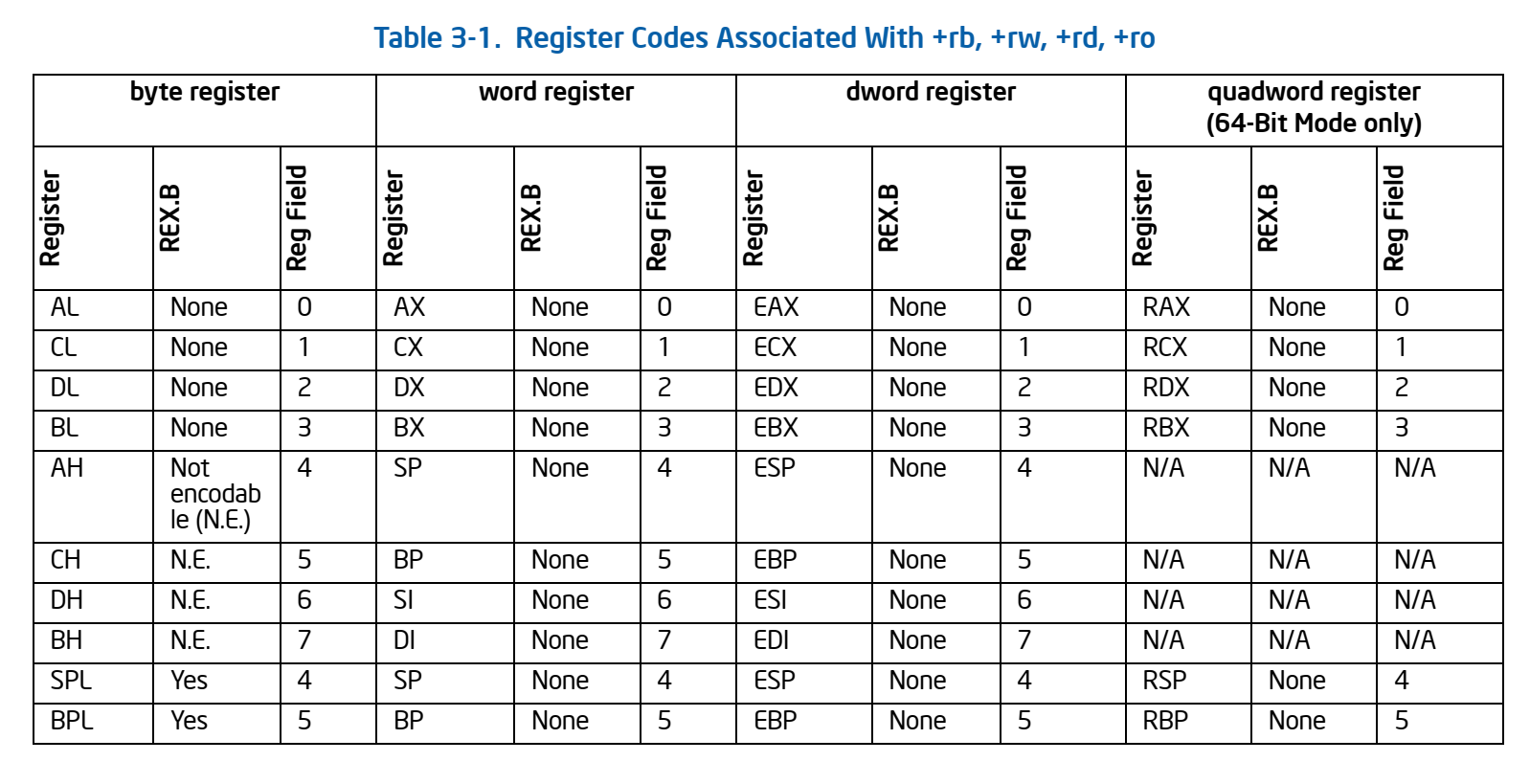
**0x0000000000408f97 ff a0 17 00 00 ac    jmpq   \*-0x53ffffe9(%rax)**

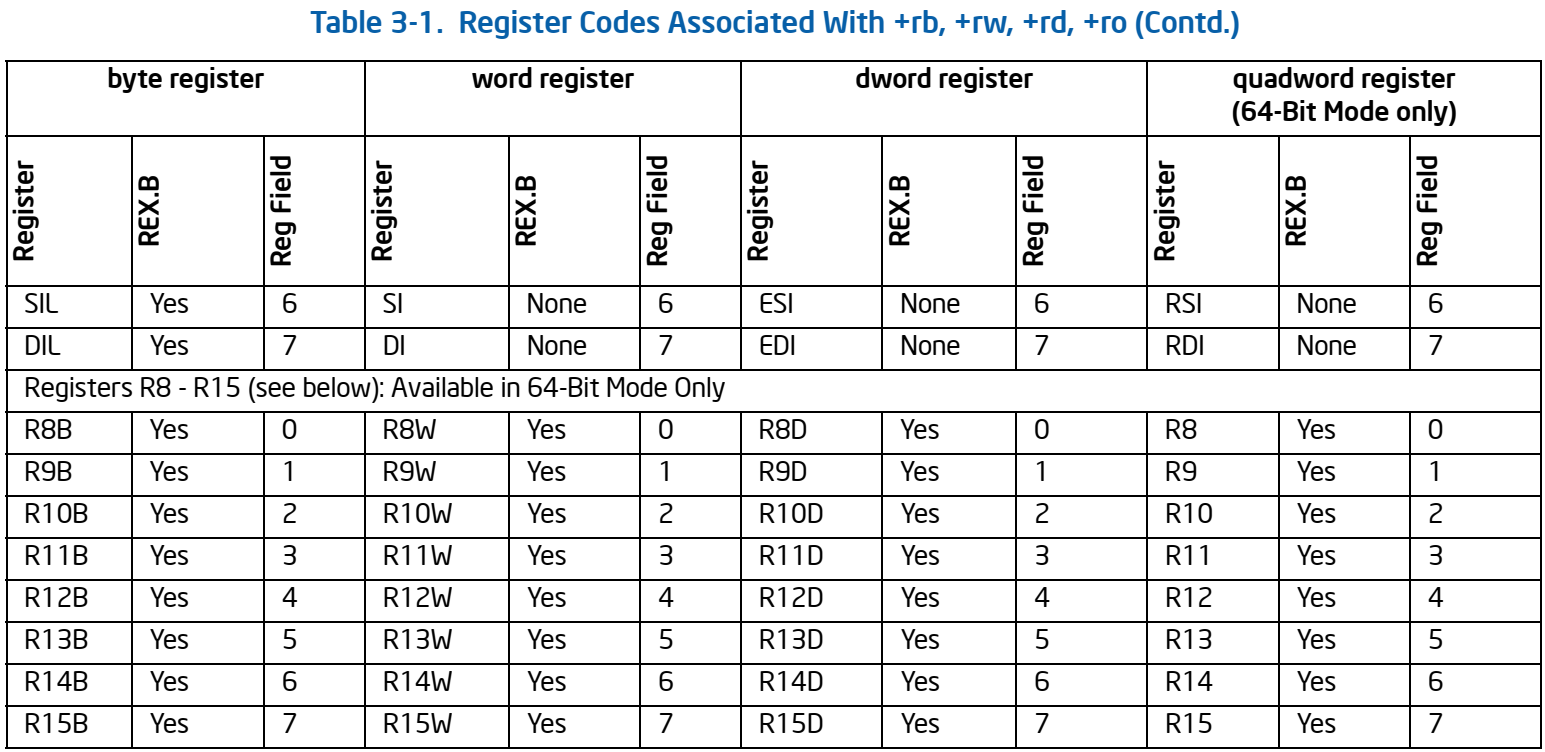
属于FF /4 JMP r/m64，跳转地址为=\*(rax + 0xac000017)，其中ModR/M=0xa0=10100000B，Mod=10B，Reg/Opcode=/4=100B，R/M=000

**0x0000000000408fbf ff a8 05 00 00 0a    ljmpq  \*0xa000005(%rax)**

属于FF /5 JMP m16:32，跳转地址为=\*(rax + 0x0a000005)，其中ModR/M=0xa8=10101000B，Mod=10B，Reg/Opcode=/5=101B，R/M=000

**九、与之相关的注册码**





**完整解释请看Intel手册Volume2 第3.1.1.1章节Opcode Column in the Instruction Summary Table (Instructions without VEX Prefix)**

　在64位模式下，使用RAX/RCX/RDX/RBX/RSP/RBP/RSI/RDI不需要REX Prefix，而使用R8 ~ R15则是必须的

**十、示例代码分析**

**10.1）示例一（x86环境）**

**012E17DD 8B 45 EC mov eax,dword ptr [ebp-14h]**

　　解析mov eax,dword ptr [ebp-14h]的机器代码：

　　　　eax为32位寄存器，[ebp-14h]是一个内存地址，所以指令应该是"MOV r32,m32"，根据第五节 MOVE-Move指令格式，符合条件的只有

https://img2018.cnblogs.com/blog/405698/202001/405698-20200130223010363-1203958053.png

　　　　通过8B /r进行分析

　　　　Instruction Prefixes：该操作码没有Instruction Prefixes。

　　　　Opcode：操作码为8B。

　　　ModR/M：/r表示操作码有ModR/M结构，且ModR/M结构的Reg/Opcode域为Reg，表示第一个操作码寄存器。

　　　第一个操作码寄存器为eax，根据第三节 ModR/M字节的32位寻址表，Reg域为000，[ebp-14h]是相对寄存器寻址，所以Mod域为01，R/M域为101，最后的ModR/M为01 000 101，即45H。

　　　　SIB：无。  
　　　　Displacement：相对ebp寄存器的偏移为-14h，-14h的源码为0001 0100，反码为1110 1011，补码为1110 1100，即EC。

　　　　Immediate：无。

　　　　最后的机器码为：**8B 45 EC**。

**10.2）示例二（x86环境）**

**012E17E3 89 45 EC mov dword ptr [ebp-14h],eax**

　　解析mov dword ptr [ebp-14h],eax的机器代码：

　　[ebp-14h]是一个内存地址，eax为32位寄存器，所以指令应该是"MOV m32,r32"，根据第五节 MOVE-Move指令格式，符合条件的只有

https://img2018.cnblogs.com/blog/405698/202001/405698-20200130223010562-832956057.png

　　通过89 /r进行分析

　　Instruction Prefixes：该操作码没有Instruction Prefixes。

　　Opcode：操作码为89H。

　　ModR/M：/r表示操作码有ModR/M结构，且ModR/M结构的Reg/Opcode域为Reg，表示第二个操作码寄存器。

　　　　第二个操作码寄存器为eax，根据第三节 ModR/M字节的32位寻址表，Reg域为000，[ebp-14h]是相对寄存器寻址，所以Mod域为01，R/M域为101，最后的ModR/M为01 000 101，即45H。

　　SIB：无。  
　　Displacement：相对ebp寄存器的偏移为-14h，-14h的源码为0001 0100，反码为1110 1011，补码为1110 1100，即EC。

　　Immediate：无。

　　最后的机器码为：**89 45 EC**。

**10.3）示例三（x86环境）**

**012E17E6 01 04 8E add dword ptr ds:[esi+ecx\*4],eax**

 　　解析add dword ptr ds:[esi+ecx\*4],eax的机器代码：

　　[esi+ecx\*4]是一个内存地址，eax为32位寄存器，所以指令应该是"ADD m32,r32"，根据第五节 ADD-Add指令格式，符合条件的只有

https://img2018.cnblogs.com/blog/405698/202001/405698-20200130223010793-2047806581.png

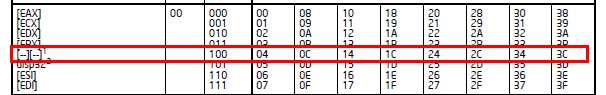
　　通过01 /r进行分析

　　Instruction Prefixes：该操作码没有Instruction Prefixes。

　　Opcode：操作码为01H。

　　ModR/M：/r表示操作码有ModR/M结构，且ModR/M结构的Reg/Opcode域为Reg，表示第二个操作码寄存器。

　　　　第二个操作码寄存器为eax，根据第三节 ModR/M字节的32位寻址表，Reg域为000，[esi+ecx\*4]是寄存器间接寻址且且使用SIB结构



　　　　所以Mod域为00，R/M域为100，最后的ModR/M为00 000 100，即04H。

　　SIB：根据[esi+ecx\*4]，SS为10，Index为001，r32为esi寄存器即110，所以SIB为10 001 110 = 8EH。  
　　Displacement：无。

　　Immediate：无。

　　最后的机器码为：**01 04 8E**。

**10.4）示例四（x64环境）**

**000000013F3D1212 48 63 4C 24 20 movsxd rcx, dword ptr [esp+20h]**

   解析movsxd rcx, dword ptr [esp+20h]的机器代码：

　　rcx为64位寄存器，[esp+20h]是一个内存地址，esp为32位寄存器，内存地址是32位地址，所以指令应该是"movsxd r64,mr32"，根据第七节 MOVSX/MOVSXD—Move with Sign-Extension指令格式，符合条件的只有

https://img2018.cnblogs.com/blog/405698/202001/405698-20200130223011239-1508523742.png

　　通过REX.W + 63 /r进行分析

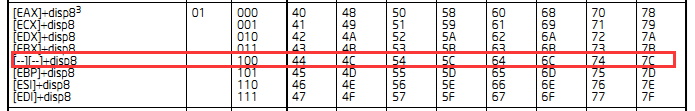
　　Legacy Prefixes：无。

　　REX Prefixes：REX.W表示操作码有REX Prefixes结构，且W位为1，则REX Prefixes为 0100 1000，即48H。

　　Opcode：操作码为63H。

　　ModR/M：/r表示操作码有ModR/M结构，且ModR/M结构的Reg/Opcode域为Reg，表示第一个操作寄存器。

　　　　第一个操作寄存器为rcx，根据第三节ModR/M字节的32位寻址表，Reg域为001，[esp+20h]是相对寄存器寻址，所以Mod域为01，表中找不到esp寄存器，所以R/M域为100。



  最后的ModR/M为01 001 100，即4CH。

　　SIB：根据[--][--]+disp8可以看出，一定有SIB结构，SIB结构为"Base + Scaled Index"，[esp+20h]没有Scaled Index，所以SS为00，Index为100，Base为esp寄存器即100，最后SIB为00 100 100 = 24H。

　　Displacement：相对esp寄存器的偏移为+20h，+20h的补码为0010 0000，即20H。

　　Immediate：无。

　　最后的机器码为：**48 63 4C 24 20**。

**参考：**

　　1）[原创]X86汇编之指令格式解析：<https://bbs.pediy.com/thread-191802.htm>

　　2）[原创]X64汇编之指令格式解析：<https://bbs.pediy.com/thread-206780.htm>

　　3）X86指令编码内幕 --- 指令格式：<https://blog.csdn.net/xfcyhuang/article/details/6228030>

　　4）Intel硬编码（一）：Opcode Map、定长指令与指令前缀：<https://blog.csdn.net/apollon_krj/article/details/77524601>

　　5）Intel硬编码（二）：不定长指令、ModR/M与SIB详解（基于P6微架构）：<https://blog.csdn.net/apollon_krj/article/details/77524601>

　　6）英特尔指令参考手册：<https://www.felixcloutier.com/x86/>

　　7）英特尔64与IA-32体系结构软件开发人员手册：<https://software.intel.com/en-us/download/intel-64-and-ia-32-architectures-sdm-combined-volumes-1-2a-2b-2c-2d-3a-3b-3c-3d-and-4>

　　8）英特尔64与IA-32体系结构优化参考手册：<https://software.intel.com/sites/default/files/managed/9e/bc/64-ia-32-architectures-optimization-manual.pdf>