Opcode解析方法。

**83**C4 04 **ADD**

每条指令，开头的字节不是前缀就是code域。

1. 先取出opcode的第一个字节，判断是code域还是前缀
2. 如果是前缀，则继续找下一个字节，再一次判断是否code域函数前缀，因为一条指令中可能有多个前缀，但绝不会超过4个，如果超过了，就是一条无效指令。
3. 如果证明就是code域的时候。
   1. 找到code域对应的指令助记符，操作数个数，操作数的类型。
   2. 如果含有Group，则用code域的值去Group中查找对应行，用指令的下一个字节，也就是ModR/M域定位Group中的列。

最终得到： ADD Ev,Ib

1. 如果存在ModR/M域， 则将code域的下一个1字节的值，去查找ModR/M表。如果存在前缀67，去16的ModR/M域表查找，如果没有，则去32位的ModR/M查找。操作数中的E和M对应着ModR/M域表的左边，G对应着ModR/M表的顶边。

第一个操作数：Ev

E => ESP/SP/AH

v => 依赖于操作数的大小（操作数指的是指令中的另一个操作数。）=> b => 字节 => AH

Ev => AH

第二个操作数：Ib

I => 立即数

b => 1个字节

Ib => 04

Add ah ,04

**A1** 78812801 **MOV**

1. A1 => code => Mov Rax,Ov => 如果是32位汇编，rax需要改成eax
2. Ov => [01288178]
3. mov eax, [01288178]

**8B 40 04**

**8B => Mov Gv,Ev**

**Gv => 40(ModR/M域) => eax**

**Ev => 40(ModR/M域) =>** [EAX]disp8 => [eax+04(**disp域**)]

Mov eax,[eax+4]

**8B 04 85 04 00 00 00**

**8B => Mov Gv,Ev**

**Gv => 04(ModR/M域) => eax**

**Ev => 04(ModR/M域) => [--][--]（SIB域：85）=> [eax\*4][00000004] => [eax\*4+4]**

**Mov eax,[eax\*4+4]**

**A3** 54812801 **MOV**

**68** 54812801 **PUSH**

**8B**0D 74812801 **MOV**

**51** **PUSH**

**68** 48812801 **PUSH**

**68** 44812801 **PUSH**

**68** 40812801 **PUSH**

**FF**15 A4922801 **CALL**