1. 何为 OPCode

在开始回答什么是 OpCode 之前，请让我先来提几个小问题。

计算机只认识 0 和 1 吗？
如果上面的回答是“是”，那么我们平时写的程序源代码是 0 和 1 吗？
如果上面的回答是“不是”，那么计算机是怎么“知道”我们的程序的意思的？

最后一个问题的答案是……？我们来举个例子，在汇编语言中：
NOP： 空指令

由于计算机只认识 0 和 1，所以，源代码“NOP”是无法直接运行的。那么该怎么办呢？
我们知道汇编在执行的时候，需要进行编译的。当编译器遇到“NOP”的时候，为了生成让计算机能运行，则生成响应的 0 和 1 指令，为了方便以十六进制数“0x90”来代替它。

在计算机科学领域中，操作码(Operation Code, OPCode) 描述机器语言指令中，指定要执行某种操作的机器码。

OPCode 在不同的场合中通常具有不同的含义，例如 PHP 虚拟机(Zend VM)、java 虚拟机(JVM)以及一些软件保护虚拟机中的最小操作单元都可以称之为 OPCode。
我们介绍的是 Intel 80x86 CPU 的 OPCode

2. 常用单字节 OPCode 概览

A -- 40~4F

opcode	asm	using
0x40	inc eax	emit(0x40)
0x41	inc ecx	emit(0x41)
0x42	inc edx	emit(0x42)
0x43	inc ebx	emit(0x43)
0x44	inc esp	emit(0x44)
0x45	inc ebp	emit(0x45)
0x46	inc esi	emit(0x46)
0x47	inc edi	emit(0x47)
0x48	dec eax	emit(0x48)
0x49	dec ecx	emit(0x49)
0x4a	dec ebx	emit(0x4a)
0x4b	dec ebx	emit(0x4b)
0x4c	dec esp	emit(0x4c)
0x4d	dec ebp	emit(0x4d)
0x4e	dec esi	emit(0x4e)
0x4f	dec edi	emit(0x4f)

B -- 50~5F

opcode	asm	using
0x50	push eax	emit(0x50)
0x51	push ecx	emit(0x51)
0x52	push edx	emit(0x52)
0x53	push ebx	emit(0x53)
0x54	push esp	emit(0x54)
0x55	push ebp	emit(0x55)
0x56	push esi	emit(0x56)
0x57	push edi	emit(0x57)
0x58	pop eax	emit(0x58)
0x59	pop ecx	emit(0x59)
0x5a	pop edx	emit(0x5a)
0x5b	pop ebx	emit(0x5b)
0x5c	pop esp	emit(0x5c)
0x5d	pop ebp	emit(0x5d)
0x5e	pop esi	emit(0x5e)
0x5f	pop edi	emit(0x5f)

C -- 70~7F

opcode	asm	using
0x70 0x12	Jo 0x12	{_emit(0x70)} {_emit(0x12)}
0x71 ...	Jno ...	... ...
0x72 ...	Jb ...	... ...
0x73 ...	Jae ...	... ...
0x74 ...	Je ...	... ...
0x75 ...	Jne ...	... ...
0x76 ...	Jbe ...	... ...
0x77 ...	Ja ...	... ...
0x78 ...	Js ...	... ...
0x79 ...	Jns ...	... ...
0x7a ...	Jp ...	... ...
0x7b ...	Jnp ...	... ...
0x7c ...	Jl ...	... ...
0x7d ...	Jge ...	... ...
0x7e ...	Jle ...	... ...
0x7f ...	Jg ...	... ...

短跳: 2 字节
第一个字节: 操作码
第二个字节: 跳转偏移

D -- 90~9F

Opcode	asm	Using
0x90	Nop/xchg eax,eax	_emit(0x90)
0x91	Xchg eax,ecx	
0x92	Xchg eax,edx	
0x93	Xchg eax,ebx	
0x94	Xchg eax,esp	
0x95	Xchg eax,ebp	
0x96	Xchg eax,esi	
0x97	Xchg eax,edi	

3. OPCode 与指令的对应关系

一个 OpCode 不只对应一个汇编指令

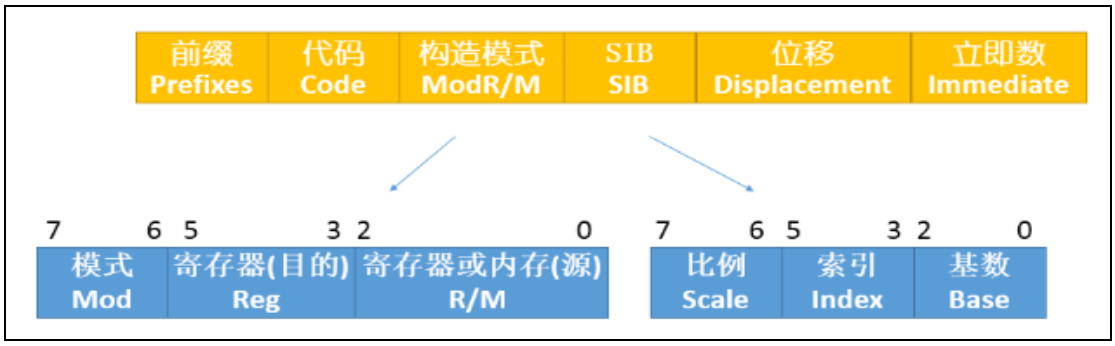
OpCode	指令
0x90	NOP
0x90	XCHG AX, AX
0x90	XCHG EAX, EAX

一个汇编指令不只对应一个 OpCode

指令	OpCode
ADD EAX, 1	0x83C001
ADD EAX, 1	0x0501000000
ADD EAX, 1	0x81C001000000

OpCode 与汇编指令的关系
一个 OpCode 不只对应一个指令。
一个指令不只对应一个 OpCode。

4. OPCode 主要数据域



以上数据域只有 Code 域是必须存在的，其他数据域视指令格式而定，或有或无

- 一个指令的长度在 1Byte~16Byte 之间
- 实际正常的最长指令是，13Byte

A --指令前缀

前缀(Prefixes)的大小为 1Byte，用于描述指令的前缀情况，他们可以被划分为 5 个集合:

66	-- 切换操作数大小
67	-- 切换地址大小
F2/F3	-- 重复操作前缀
2E/36/3E/26/64/65	-- 修改默认段
F0	-- 锁定前缀

所以指令独此一份，不可能为其他机器码

注意:

a. "切换"的意思是将其在两种状态间来回切换，而并非特指某种状态
b. 将默认值修改为其他段的操作称之为"修改默认段"
c. 一个 OpCode 可能会有几个 Prefixes
d. 如果有多个 Prefixes，那么它们的顺序可能打乱
e. 如果 Prefixes 不能对随它之后的 OpCode 起作用，那么它就会被忽略

出现特定操作码时用作补充说明，图中的冒号前的 64 就是指令前缀

```
64:FF30          push dword ptr fs:[eax]
64:8920          mov dword ptr fs:[eax],esp
80BB 94000000    cmp byte ptr ds:[ebx+0x94],0x0
```

切换操作数大小

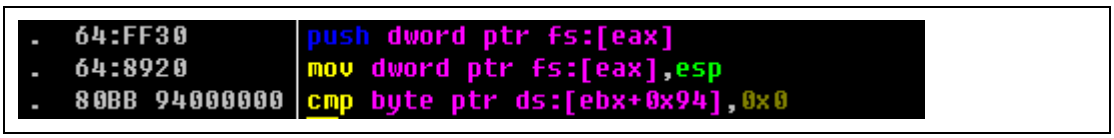
40 INC EAX

66 40 INC AX

切换顺序: 从大到小

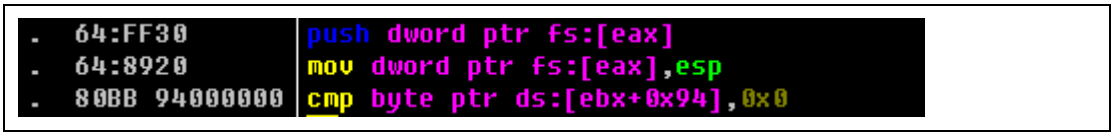
B --操作码

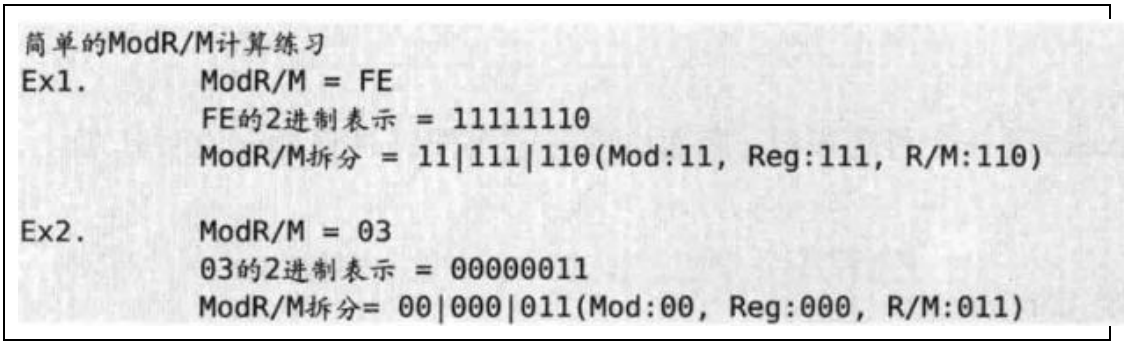
实际的指令，如图中的 FF、89、80 都是操作码



C -- ModR/M

辅助说明操作码的操作数（操作数的个数、种类[寄存器、内存地址、常量]），图中的 30、20、BB





3D - ModR/M：转为二进制 00 111 101

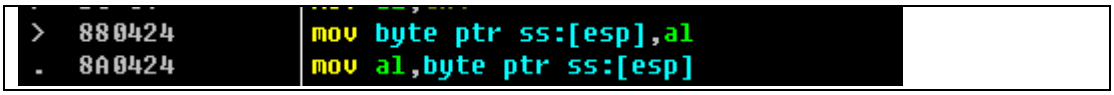
模式(Mod)段 ： 2 位 00

寄存器(Reg)段 ： 3 位 111

寄存器(R/M)段 ： 3 位 101

D – SIB

用来辅助说明 ModR/M，辅助寻址，图中的两个 24 都是 SIB。操作码的操作数为内存地址时，需要与 ModR/M 一起使用



```
*简单的SIB计算练习
Ex1.        SIB = 24
            24的2进制表示 = 00100100
            SIB拆分 = 00|100|100(Scale:00, Index:100, Base:100)

Ex2.        SIB = 01
            01的2进制表示 = 00000001
            SIB拆分 = 00|000|001(Scale:00, Index:000, Base:001)
```

24 - SIB：转为二进制 00 100 100

比例(Scale)段 ： 2 位 00

索引(Index)段 ： 3 位 100

基数(Base)段 ： 3 位 100

E -- 位移

操作码的操作数为内存地址时，用来表示位移操作，图中的 D0A44500 为位移，图中小端序排列

```
.  8B1D D0A44500    mov ebx,dword ptr ds:[0x45A4D0]
```

F -- 立即数

操作码的操作数为常量时，该常量就被称为立即数，图中的 02000000 为立即数

```
.  BA 02000000      mov edx,0x2
```

5. 练习

```
1 00F63689 : E8 C5F9FFFF
 2 00F6368E : 6A 58
 3 00F63690 : 68 A037F600
 4 00F63695 : E8 72040000
 5 00F6369A : 33DB
 6 00F6369C : 895D E4
 7 00F6369F : 895D FC
 8 00F636A2 : 8D45 98
 9 00F636A5 : 50
10 00F636A6 : FF15 FC10F600
11 00F636AC : C745 FC FEFFFFFF
12 00F636B3 : C745 FC 01000000
13 00F636BA : 64:A1 18000000
```

```
14 00F636C0 : 8B70 04
15 00F636C3 : BF 5CC2F600
16 00F636C8 : 6A 00
17 00F636CA : 56
18 00F636C0 : 57
```

操作数类型的含义

E8 C5F9FFFF

E8 : near callf64 Jz

Jz 操作数，看附件处可知是指相对偏移量，FFFFF9C5 为 call 目的地址的偏移量（向上），所以偏移为-63B（FFFFF9C5 是一个负数，以 16 进制的补码形式显示，00000000-FFFFF9C5=-63B）,再加上指令的 5 个字节为-636，指令所在地址为 00F63689，所以汇编代码为：call 00F63053

6A 58

6A : PUSHd64 Ib

Ib 是一个字节的立即数，所以汇编代码为：push 0x58

68 A037F600

68 ：PUSHd64 Iz

Iz 是立即数（4 个字节），所以汇编代码为：push 00F637A0

E8 72040000

E8 : near callf64 Jz

和上面一样，但是这里的偏移量是 00000472（向下偏移），加上指令的 5 个字节为 00000477，指令所在地址为 00F63695，汇编代码为：call 00F63B0C

33DB

33 : XOR Gv,Ev

Gv 表示寄存器，Ev 为寄存器（DB 在 C0~FF 之间）

DB 为 ModR/M

DB 的二进制形式 = 11011011

拆分 ModR/M = 11|011|011(Mod:11,Reg:011,R/M:011)

在 ModR/M 表里找到寄存器为 ebx，ebx

所以汇编代码为：xor ebx,ebx

895D E4

89 : MOV Ev,Gv

Ev 为内存地址（5D 在 00~BF 之间），Gv 表示寄存器

5D 为 ModR/M

二进制形式 = 01011101

拆分 ModR/M = 01|011|101(Mod:01,Reg:011,R/M:101)

命令可得：MOV [EBP]+disp8, EBX

disp8 指 1 个字节的偏移，E4 可得-1c

所以汇编代码为：mov dword ptr ss:[ebp-0x1C],ebx

（ebp+或 ebp-地址，一般指堆栈里，所以用 dword ptr ss:[]这样的形式）

895D FC

89 ：MOV Ev，Gv

同上，所以指令为：mov dword ptr ss:[ebp-0x04],ebx

8D45 98

8D ：LEA Gv, M

Gv 表示寄存器，M 表示内存地址

ModR/M：45

二进制形式 = 01000101

拆分 ModR/M = 01|000|101(Mod:01,Reg:000,R/M:101)

命令可得：LEA EAX， [EBP]+disp8

98 的补码为-68（怎么看出补码，一个字节范围-128~127，如果是大于 7F 即 127，就是补码）

所以指令为：lea eax, dword ptr ss:[ebp-0x68]

50

50 ：pushd64 rAX/r8

就是 push eax

FF15 FC10F600

FF ：INC/DEC Grp5^1A

ModR/M：15

二进制形式 = 00010101

拆分 ModR/M = 00|010|101(Mod:00,Reg:010,R/M:101)

所以 Grp5 为 near callf64 Ev

Ev = [disp32]，Ev 为内存地址（15 在 00~BF 之间)

指令为 call 00F610FC（根据后面的 group 为主，前面的 INC/DEC 推测因为 Reg 不是 000 或 001，所有没有用到）

C745 FC FEFFFFFF

C7 ：Grp11^1A-MOV Ev, Iz

ModR/M：45

二进制形式 = 01000101

拆分 ModR/M = 01|000|101(Mod:01,Reg:000,R/M:101)

Grp11 为 MOV Ev, Iz

Ev 是内存地址（45 在 00~BF 之间)，Iz 是立即数

根据 Mod=01，R/M=101，指令化为：MOV [EBP]+disp8,Iz

最后 FC(-0x04),FFFFFFFE(-0x02)

指令为：mov dword ptr ss:[ebp-0x04], -0x02

C745 FC 01000000

同上，指令为：mov dword ptr ss:[ebp-0x04], 00000001

64:A1 18000000

64 为前缀，可知 SEG=FS

A1 ：MOV rAX, Ov

Ov 存放 fs:[]

所有指令为 mov eax, fs:[0x18]

8B70 04

8D ：LEA Gv，M

Gv 表示寄存器，M 表示内存地址

ModR/M：70

二进制形式 = 01110000

拆分 ModR/M = 01|110|000(Mod:01,Reg:110,R/M:000)

LEA ESI，[EAX]+disp8

指令为：lea esi，dword ptr ss:[eax+0x04]

BF 5CC2F600

BF : MOV rDI/r15, Iv

rDI/r15 是寄存器 edi

Iv 是一个立即数：00F6C25C

指令为：mov edi, 00F6C25C

6A 00

6A : PUSHd64 Ib

Ib 是一个字节大小的立即数，所以

指令为：push 00

56

56 ：push esi

57

57 ：push edi

结果

```
1 00F63689 : call 00F63053
```

```
 2 00F6368E : push 0x58
 3 00F63690 : push 00F637A0
 4 00F63695 : call 00F63B0C
 5 00F6369A : xor ebx,ebx
 6 00F6369C : mov dword ptr ss:[ebp-0x1C],ebx
 7 00F6369F : mov dword ptr ss:[ebp-0x04],ebx
 8 00F636A2 : lea eax, dword ptr ss:[ebp-0x68]
 9 00F636A5 : push eax
10 00F636A6 : call 00F610FC
11 00F636AC : mov dword ptr ss:[ebp-0x04], -0x02
12 00F636B3 : mov dword ptr ss:[ebp-0x04], 00000001
13 00F636BA : mov eax, fs:[0x18]
14 00F636C0 : lea esi， dword ptr ss:[eax+0x04]
15 00F636C3 : mov edi, 00F6C25C
16 00F636C8 : push 00
17 00F636CA : push esi
18 00F636C0 : push edi
```

附件

操作数类型的含义

常用寻址方法

E ： 内存地址(00~BF)或寄存器（C0~FF）
A ModR/M byte follows the opcode and specifies the operand. The operand is either a general-purpose register or a menory address
ModR/M 字节在操作码后面并指定操作数。操作数是一个通用寄存器或一个内存地址

G ： 寄存器
The reg field of the ModR/M byte selects a general register(for example, AX(000))
ModR/M 字节的寄存器字段选择通用寄存器（例如：AX(000)）

I ： 立即数
Immediate data : the operand value is encoded in subsequent bytes of the instruction
立即数：操作数的值在指令的后续字节中编码

J ： 相对偏移
The instruction contains a relative offset to the instruction pointer register(for example, JMP(0E9), LOOP)
这个指令包括了指令指针寄存器的相对偏移量（例如：JMP（0E9），LOOP）

M ： 内存地址
The ModR/M byte may refer only to menory(for example, BOUND, LES, LDS, LSS, LFS, LGS, CMPXCHG8B)
ModR/M 字节可能只涉及内存（例如：BOUND，LES，LDS，LSS，LFS，LGS，CMPXCHG8B）

X ： 内存地址
Memory addressed by the DS:rSI register pair(for example, MOVS, CMPS, OUTS, or LODS)
由 DS:rSI 寄存器对寻址的内存地址（例如：MOVS，CMPS，OUTS，LODS）

Y ： 内存地址
Memory addressed by the ES:rDI register pair(for example, MOVS,CMPS,INS,STOS,or SCAS)
由 ES:rDI 寄存器对寻址的内存地址（例如：MOVS，CMPS，INS，SIOS 或 SCAS）

操作数类型

b
Byte,regardless of operand-size attribute
字节，不管操作数大小属性如何

d
Doubleword, regardless of operand-size attribute
双字节，不管操作数大小属性如何

v
Word, doubleword or quadword(in 64-bit mode), depengding on operand-size attribute
字，双字或四字（64 位模式），取决于操作数大小

z
Word for 16-bit operand-size or doubleword for 32 or 64-bit operand-size
16 位操作数大小的字，32 位或 64 位大小的双字