滴水信息

滴水逆向培训

基础教程

昆山滴水信息技术有限公司

（内部学习资料）

地址：昆山市巴城镇学院路 88 号
邮编：215311
TEL：0512-57882866
官网：www.dtdishui.com
论坛：www.dtdebug.com
EMAIL：kunshandishui@163.COM

前言

在写这篇前言之前，首先非常感谢多年以来一直关心和支持滴水的朋友们，同时也非常感谢那些不停的鞭策我们勇往直前的批评者。在你们的掌声与谩骂声中我们一路走来。

2008 年滴水公司成立之日起，我们就面临着一个非常严峻的现实，巨大的工作量无人可以分担。既招不到合适的程序员，也找不到愿意同行的伙伴。在孤独与寂寞中我们慢慢的爬行。。。。。。

国内有不计其数的计算机专业毕业生，但是真正的程序员却少之又少。大量院校开办计算机专业，而真正酷爱编程的学习者却无处求学。只有少数人通过自学成为真正合格的程序员。业内的很多人其实都知道原因。大学缺乏合格的计算机专业老师，只会照本宣科、没有工程实践的老师永远也教不出合格的程序员。而真正经验丰富、适合做老师的程序员又没有教书育人的机会。现实需要有人能够改变这种不合理的现状。

2011 年经过深思与熟虑，滴水决定暂时放下产品开发，开办逆向培训。我们要做一些有意义的事情。即使我们是大海中的一滴水，也可以荡起一点点的涟漪。知行合一，立刻行动才是我们所需要的！

首先介绍一下滴水逆向培训的主讲老师-唐老师。2000 年毕业于西北一所不知名的大学，贸易经济专业毕业—似乎资深的程序员大多非计算机专业出身。有着 10 年以上的编程经验，取得过计算机软件发明专利，业内公认的杰出程序员。唐老师是滴水硬件调试器和 VT 调试器的开发者，成功逆向虚拟机 VMWare、加壳软件 Themida 和 VMProtect，并即将完成滴水动态变形壳的开发。

接下来说明一下本教程的编写历程。滴水逆向培训基础教程是唐老师自学成才的学习历程以及 10 多年软件开发经验的总结，并经过滴水逆向培训前几期的讲课实践整理编辑而成。以此让更多无法参加实地培训又渴望学习的人一起分享我们的经验。

在此需要说明的是此教程不同于我们以往常见的教材。为了尽可能的保持唐老师教学的原貌，本教程采用语录+章节体的风格编辑整理而成。先引述唐老师的讲课内容，然后再添加学生实际学习中整理、消化吸收的历程。为读者尽可能的创造一种身临其境、置身其中的学习氛围，共同探讨和学习本教程的内容。由于个人天赋的差异，以及文字表达的局限性，每个读者可能都会有不同的理解，甚至有些比较深奥的地方难以理解。这都是正常的，即使

参加实地培训的学员相互之间也存在着巨大的差距。在此只能靠每个读者细心的领悟。同时，由于编写的时间过于仓促，教程本身可能存在诸多错误，希望读者多多批评指正，我们将在后续版本中不断的加以纠正和完善。

最后，我们需要特别强调学习本教程的注意事项：

适合的读者：本教程适合于任何有志于从事编程的读者，包括零基础在内。零基础可能会比较吃力，但有志者事竟成！

教学的目的：帮助零基础的人跨过计算机门槛，为有志于从事计算机编程工作的读者打下坚实的基础。培养独立的自学能力。

学习效果：打下坚实的基础知识，可以独立学习任何自己感兴趣的东西。

学习的前提条件：一是数学要好；数学知识的掌握程度反映一个人的逻辑思维能力，这是学习编程的必备条件，否则会在后续的学习中非常吃力，难以达到预期的学习效果。如果你初中数学优异基本就可以证明具备一定的逻辑思维能力了。二是有兴趣；兴趣是最好的老师。如果这不是你终身希望做的事，学习本教程将是一件非常枯燥无味的事，最终会放弃的。三是肯用功；即使你的智商超过爱因斯坦也不能确保你一定成绩优异，天才+勤奋才是成功的秘诀。补充说明一点，学习编程和英语水平没有必然联系。因为编程涉及的英语单词有限，看的次数多了自然记得。

学习方法：勤学+苦练。练习到手都麻了为止。反复的练习是最佳的捷径，偷懒是你最大的敌人。任何人概莫能外，切记！我们将会在每一章节安排必要的练习，千万不可以偷懒。

教学内容及顺序：几乎所有教材的知识内容大致都是相同的，但是最后学习的效果却有天壤之别，很大一部分原因在于学习的顺序。正确的教学顺序才能让我们在计算机浩如烟海的知识体系中找到正确的方向，循序渐进的消化和吸收才可以真正掌握所学习的内容。大学计算机教育的失败很大一部分原因在于教学内容和顺序的安排错误。表面上所有的内容都已经学过了，但是学完之后全部都忘了，一无所获。过多的学习内容只会让学生望而却步、疲于应付。在没有很好的消化吸收前段学习内容的情况下，继续填鸭式的学习新的内容，割裂了各个知识点之间的联系，最终导致学习失败。所以我们要求，在没有充分掌握前一段知识的情况下，不要继续学习后续的内容。希望能够引以为戒。本教程的教学内容抛弃了知识体系中大量多余的内容，只讲述必要的内容，并进行有效的穿插，帮助读者把每个知识点前后联系起来，并最终牢牢的记住。这是我们所坚持的正确的教学安排，特意为之。学习完本教程，打下一定的基础之后，就可以根据自己的爱好和工作要求自学完成本教程之外暂时没有教授的知识了。所以教程没有涉及过多的应用知识，这是我们有意安排的。学完本教程，打

好基础之后，其他的应用知识可以自学完成。基础决定一切。挖多深的地基才能盖多高的楼。这就是我们为什么从汇编开始的原因。只有从底层的汇编开始学起，才是唯一正确的方法。汇编基础打好之后，其他知识就会学的越来越轻松。

此外，由于时间的限制，我们将会在编辑教程中引用其他优秀教材的部分内容，在此向他们表示诚挚的敬意！

最后再次感谢支持和鞭策滴水的朋友们，感谢滴水员工胡雨和陶捷为编写本教程所付出的辛勤工作。我们将在实际教学中不断完善本教程，对于一些自学能力相对较弱的读者，欢迎在情况允许的情况下报名参加滴水逆向实地培训。如果在学习本教程过程中遇到难以理解的问题也可以登录滴水论坛与大家一起探讨，我们将尽可能的给予解答。滴水官网地址：www.dtdishui.com;论坛地址：www.dtdebug.com；谢谢！

昆山滴水信息技术有限公司

2013 年 1 月 8 日

《滴水逆向培训基础教程》电子版仅供参考，书面教程更为详尽。

目录

第二章 寄存器与汇编指令

引言

想要了解计算机，首先要了解的便是 cpu，cpu 是计算机最核心的部件，因为计算机的所有指令都是由 cpu 处理，而 cpu 的核心部件之一是寄存器。这一章我们就来认识一下寄存器以及寄存器是如何工作的。

本章必须要掌握的知识点：

1．8、16、32 位通用寄存器

2．寄存器与内存的区别

3．汇编语言的基础指令

本章常犯的错误：

1．内存的存储格式

2．溢出标志位（OF）的理解

3．pop 与 push 指令的理解

2.1 通用寄存器

本节主要内容：

1．8/16/32 位通用寄存器

2．汇编指令

老唐语录：

计算机最经典的指令就是移动指令：mov ecx，eax；

主要记住这 8 个寄存器：

eax

ecx

edx

ebx

esp

ebp

esi

edi

mov 指令可以任意移动这 8 个寄存器。

在 mov ecx，eax 中，后面的是源，前面的是目标，中间是逗号，不区分大小写。寄存器间相互移动。

如果计算机只有移动指令的话，那么什么事也干不了。

mov 是操作码，两个寄存器是操作数，操作码除了 mov 之外还有很多，你可以替换：加，ADD；减，SUB；与，AND；或，OR；异或，XOR；非，NOT。

比如：add eax,ecx。

当然了，操作码不仅仅只有这几个，还有很多很多，只要搞清楚操作数是任意两个寄存器，当然可以是同一个寄存器。

比如 or esi，esi：将 esi “或” esi 并将结果赋给 esi。操作数除了寄存器之外，还可以是一个数，只要保证是 32 位即可。

8 个寄存器是分段的。比如算盘，我们可以只用一半，或者四分之一。

eax 可以分成四部分：eax 共 0-31，其中 0-7 位叫做 AL，8-15 位叫做 AH。整个 0-15 位又称为 AX。AL：low；AH：high。ecx，edx，ebx 也是一样的，如图 2-1。

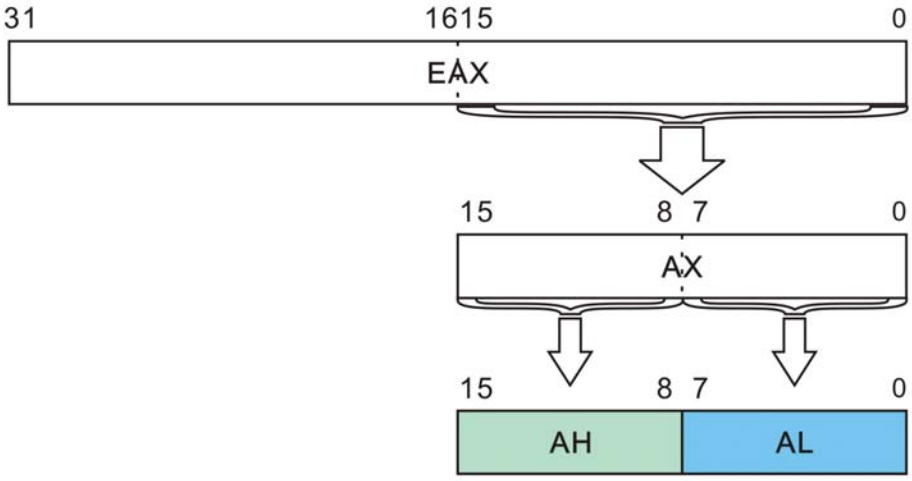

图 2-1：EAX 寄存器

以上四个寄存器都可以分成三段。

ESP,EBP,ESI,EDI，这四个寄存器只分成两段，比如 ESP 整体 0-31 位称为 ESP，0-15 位称为 SP。

通过以上，我们可以看出，32 位的寄存器有 8 个：EAX，ECX，EDX，EBX，ESP，EBP，ESI，EDI，每个寄存器都有一个编号：0 号，1 号，2 号，3 号……还有 16 位的寄存器：AX，CX，DX，BX，SP，BP，SI，DI。也是一样的：0 号，1 号，2 号，3 号…… 如表 2-1：

表 2-1：寄存器编号

寄存器			编号（二进制）	编号（十进制）
32 位	16 位	8 位		
EAX	AX	AL	000	0

ECX	CX	CL	001	1
EDX	DX	DL	010	2
EBX	BX	BL	011	3
ESP	SP	AH	100	4
EBP	BP	CH	101	5
ESI	SI	DH	110	6
EDI	DI	BH	111	7

32 位寄存器有自己的编号，16 位寄存器也有属于自己的独立的编号。当然，他们是重叠的，当改变了 32 位的寄存器，相应的 16 位寄存器也会跟着改变。

同样，也有 8 位的寄存器，第 0 号 AL，第 1 号，CL，DL，BL，AH，CH，DH，BH。顺序不能乱。当然还有两个寄存器：EIP 和 EFLAGS（又称为 EFL），8 号和 9 号寄存器，EIP 有 16 位，叫做 IP。EFL 的 16 位称为 FL。这两个寄存器使用相对较少。

练习：

除了 EIP 和 EFL 之外，其他寄存器都是可用的，使用 MOV，ADD，SUB……等指令做练习，比如 XOR AL，BH，后面随便跟两个寄存器，用逗号隔开，当然，前后宽度要一样。而不可以是 XOR AL，BX。后面的寄存器可以改成立即数。

课后理解：

通用寄存器即 cpu 常用的寄存器。Intel 手册给出了通用寄存器的功能：通用目的寄存器（General-Purpose Registers）主要实现逻辑和算术运算、地址计算和内存指针。

通用寄存器结构见图 2-2：

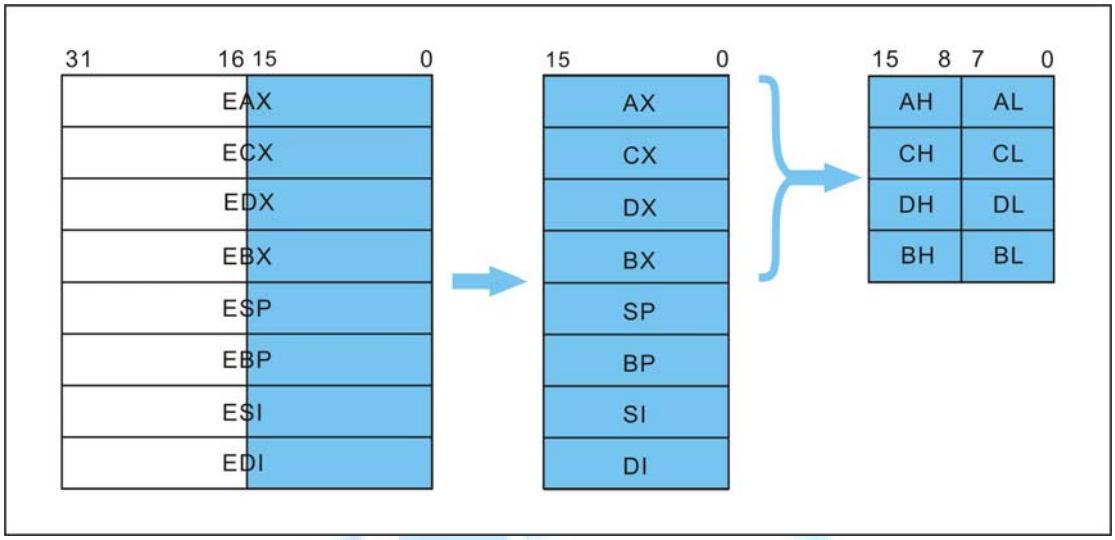

图 2-2：通用寄存器结构

为什么上图看上去比较奇怪，有的寄存器被分成一块一块？其实都是历史遗留问题，在早期的技术还没有现在成熟时，8 个通用寄存器宽度是 8 位（这里指的是二进制的 8 位）：AL，AH，CL，CH，DL，DH，BL，BH。后来 cpu 技术逐渐发展，由 8 位发展成 16 位，于是寄存器由 8 位演变成 16 位：AX，CX，DX，BX，SP，BP，SI，DI。但是为了兼容前面的 8 位技术，将之前的 8 个 8 位寄存器封装到 AX，CX，DX，BX 寄存器中。技术发展的脚步不会停歇的，之后将 16 位寄存器都扩充了一倍，于是 32 位寄存器出现了，当然这里只截至 32 位，64 位和 128 位寄存器留给大家思考。

32 位通用目的寄存器的指定用途如下：

- EAX：累加器（Accumulator）
- ECX：计数（Counter）
- EDX：I/O 指针
- EBX：DS 段的数据指针
- ESP：堆栈（Stack）指针
- EBP：SS 段的数据指针

• ESI：字符串操作的源（Source）指针；SS 段的数据指针

• EDI：字符串操作的目标（Destination）指针；ES 段的数据指针

以上只是让大家对寄存器的缩写的含义了解一下，毕竟使用一样东西，不明白其中的道理，实在不高明，而且这东西是老外发明的，按中式逻辑很难猜出来。

课后疑问：

问题：EIP 在不断变化，我们不但要有命令去操作这些寄存器，还需要区分当前用的是哪一条命令，下一次再用哪一条命令。

回答：**cpu 具有判断指令长度和预处理指令的功能。**

为什么你写的寄存器顺序是 EAX，ECX，EDX，EBX……而不是 EAX，EBX，ECX，EDX……，不是更容易记住?

回答：其实我也想，可是 Intel 的技术员不这么认为，他们就是这么定义寄存器顺序的。**了解寄存器的顺序和编号，对后面将要介绍的汇编指令和硬编码尤为重要。**

课后总结：

汇编就是在“寄存器与寄存器”或者“寄存器与内存”之间来回移动数据。

课后练习：

32，16，8 位寄存器用 mov add AND sub or xor not 演算

2.2 内存

本节主要内容：

1．寄存器与内存的区别

2．寄存器,内存

3．内存修改

老唐语录：

快速算盘叫做寄存器。慢速的称为内存。其实他们的结构差不多，都是定宽的，最重要的一点，寄存器速度非常快，价格非常昂贵，所以在 CPU 内部。做的数量也很有限。常用的只做了 8 个：EAX,ECX,EDX,EBX,ESP,EBP,ESI,EDI。内存速度慢，很便宜，所以数量做的非常庞大。寄存器做的数量非常少，就可以为每个寄存器取个名字。而内存数量太庞大了，没办法给每一个都取上名字，所以只能编号。最后重要的是内存中寄存器的编号是 32 位的。这也是我们现在的计算机叫 32 位计算机的主要原因。如果按寄存器的宽度是 32 位的话，是不对的，因为还有很多寄存器是大于 32 位的。我们通用的 8 个是 32 位的。而内存地址是固定的 32 位。以前的计算机之所以称为 16 位计算机，就是因为他的内存单元编号是 16 位的。

我们学过的指令 MOV，能操作寄存器和内存。

所以说，汇编可以用一句话概括：汇编就是在寄存器和寄存器或寄存器和内存之间来回移动数据。就是指数据在内存和寄存器间来回流动，流动的多了，代表程序很复杂，比如 office 等一些大型软件。

mov eax，0x401000//给 eax 初始化

MOV 也可以改成 ADD（加），SUB（减），XOR（异或），OR（或），AND（与）。

后面的操作数为源操作数，前面的操作数为目标操作数，最开始是操作码。

其实 ADD……这些也是移动指令：把源加上目标然后移动到目标操作数里面去。

前面表示操作方式，后面表示寄存器和寄存器或寄存器和内存，但是只能出现一个内存。比如：ds:[]里面写个数，表示内存单元。ptr 指的是指针 point。word 是两个字节，还有 byte 是一个字节，dword 是四个字节。

写法：byte/word/dword ptr ds:[32 位的数]。

其实[]里面不但可以写具体的一个数，还可以写某一个寄存器的值。

方括号的通用格式：reg+reg*数+立即数，只要这个值算出来指向哪一个内存单元，就是那个内存单元（现在先记住，不要管为什么，后面会讲）。

第一个寄存器叫做 BASE 寄存器（8 个寄存器都可以），第二个寄存器是 INDEX 也是 8 个寄存器之一，后面乘一个数 scale（1，2，4 或 8，也就是 2 的 0 次方，2 的 1 次方，2 的 2 次方，2 的 3 次方），后面再加一个数（DISP）。

BASE+INDEX*（1，2，4，8）+DISP

可以是有以下五种组合

BASE

INDEX*（1，2，4，8）

DISP

BASE+INDEX×（1，2，4，8）

BASE+INDEX×（1，2，4，8）+DISP

每一种组合都可以。

我们只有把最简单的东西用熟了，才能熟能生巧。

练习：

mov eax,ds:[0x401000+8*eax+eax];这个可以执行吗？

cpu 实现了将地址赋给寄存器：LEA 指令把方括号里面内存的编号给目标寄存器：load effective address。

当一个值不好确定宽度时，使用 dword ptr 或者 word ptr，byte ptr，源可以是内存或寄存器，也可以是立即数，但是目标不能是立即数。两方都可以是内存，但是内存只能在一个地方出现。两边的数据宽度要一样。所以指令是由操作码和操作数组成，操作码是表明我想干什么。复杂的指令也是由简单的指令组合而成。

课后理解：

内存同样由许多寄存器组成，但是此寄存器非彼寄存器，速度不及通用寄存器的几十分之一，宽度为 8 位。见图 2-3：

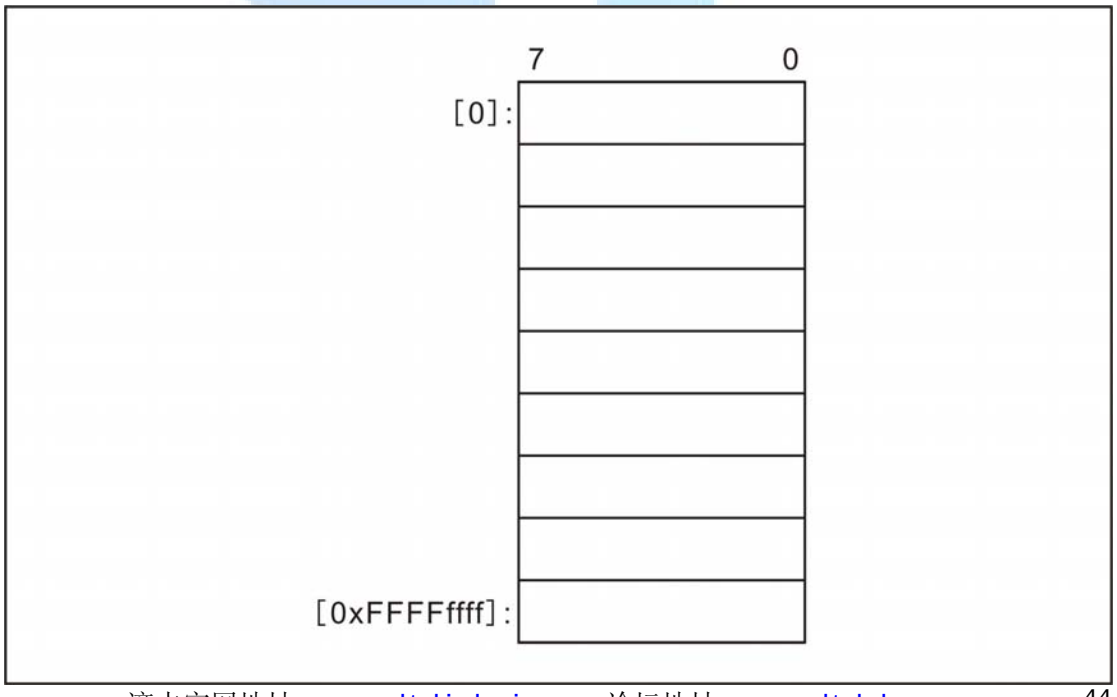

图 2-3：内存格式

如图 2-3，为了区别内存和 cpu 内部的寄存器，我们将内存中的寄存器打上“[]”，专业术语称之为“地址”。图中的内存大小从 0 到 0xFFFFffff,也就是说有 0xFFFFffff 个内存寄存器存在，内存地址从 0-0xFFFFffff。

注：此处 0xFFFFffff 写法为何不写成 oxFFFFFFFF？

十六进制不区分大小写，这样写是为了便于识别，我们可以很清楚的看出有 8 个 F。

内存地址的用处是什么？

当用户运行程序时，cpu 需要不停地去从存储区取代码和数据，这样非常耗时，于是 cpu 先将可能用到的代码和数据从存储区全部放入内存，再从内存中取数据和代码。看似多了一个过程，但是从内存中获取比存储区快得多，所以节省了很多时间。

我们如何使用内存地址呢？

我们通过实例来了解。

将 32 位数 0x12345678 存入内存中的 0x12FFB8 地址处：

说明：我们从图 2-4 可以看出：数值的高位存储在内存地址中的高位。

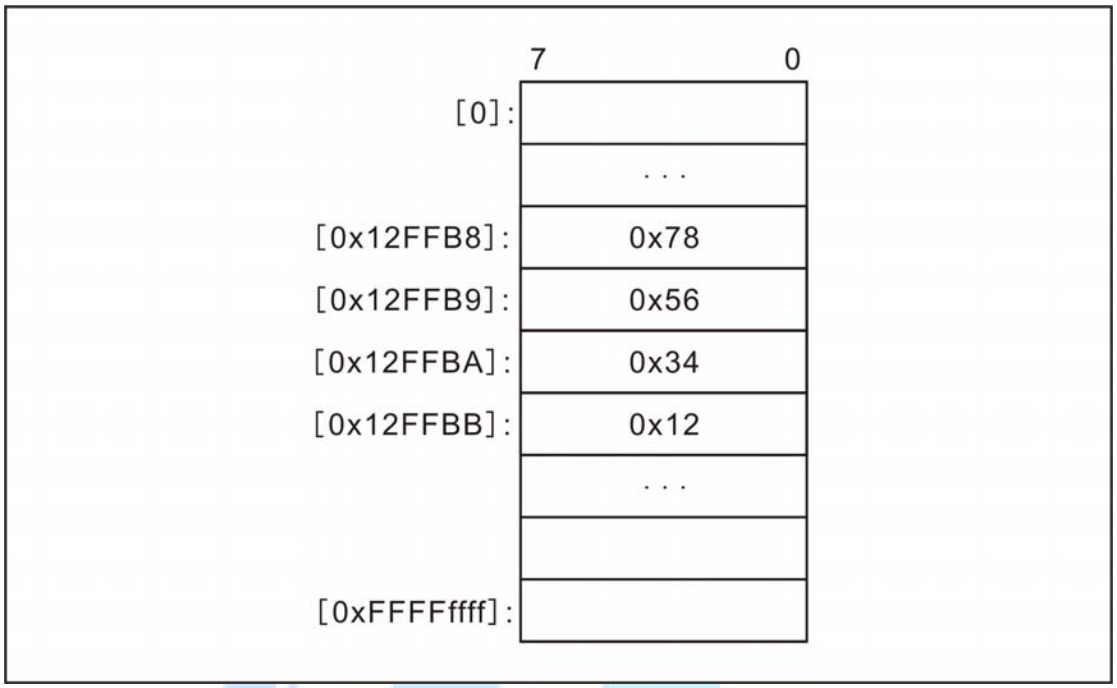

图 2-4：内存存储方式

内存地址的表示方法有哪些?

内存地址的表示方法有很多，除了上图中的表示方法外，还有其他四种表示方法。

以下是内存地址表示方法的组成成员：

- **位移**（Displacement） - 8 位、16 位或 32 位值
- **基**（Base） - 通用目的寄存器
- **索引**（Index） - 除 ESP 外的通用目的寄存器
- **比例因子**（Scale Factor） - 1，2，4 或 8

下列五种地址模式为常用组合（图 2-3）：

- 位移
- 基
- 基 + 位移

•（索引×比例因子）+ 位移

• 基 + （索引×比例因子）+ 位移

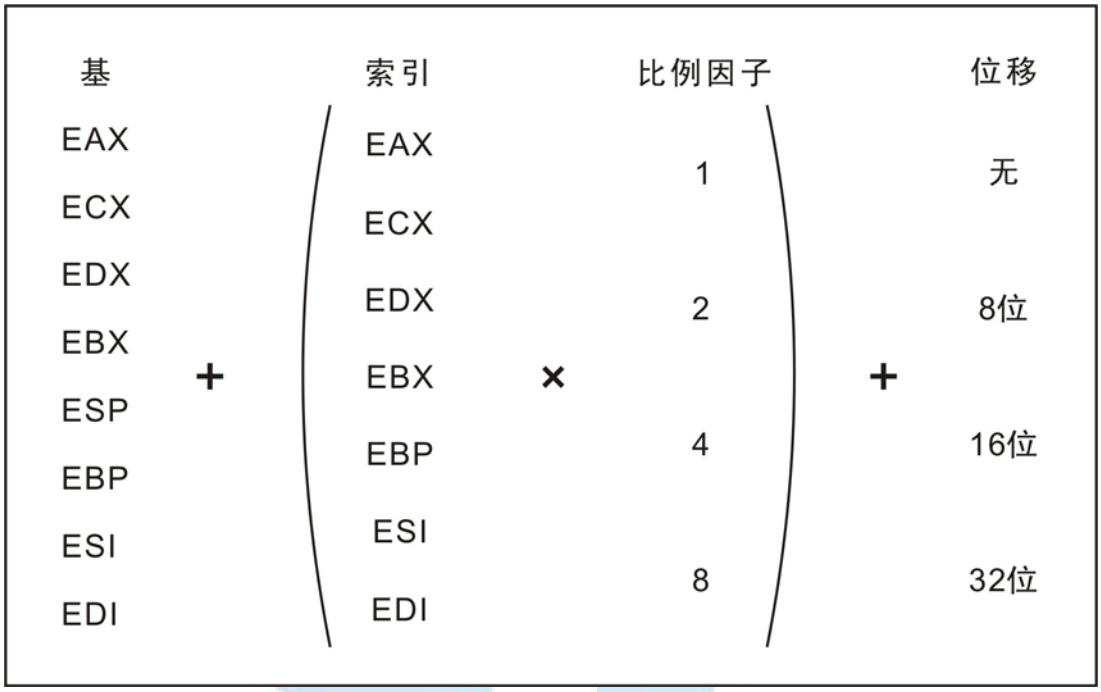

图 2-5：偏移（有效地址）计算

为什么只有五种表示方法，而且比例因子还有限制？

极有可能的原因是（猜测）：计算机只识别机器语言，所以我们要将内存地址的表示方法翻译成机器语言才能得到执行。组合越多，翻译起来越麻烦，cpu 的技术员们只好订个规矩：只能使用五种表示方法，否则一律不识别。

练习：

用实例表示以上五种组合方式。

1.[0x234791]//vc6 不支持

2.[reg]

3.[reg+0x10234]

4.[reg+reg*{1,2,4,8}]

5.[reg+reg*{1,2,4,8}+0x1234]

注：reg 表示通用寄存器。

课后疑问：

1. scale 可以是其他值吗？

回答：不可以。

2. 如果算出的内存地址结果超过 32 位会怎样？

回答：如果结果超过 32 位溢出，则计算机只取 32 位。

课后总结：

内存的通用格式：reg+reg*数+立即数

课后练习：

练习复杂地址表达形式的操作如：lea [eax +eax*0x02+0x12548],eax