第1天　从计算机结构到汇编程序入门

- 先动手操作
- 究竟做了些什么
- 初次体验汇编程序
- 加工润色

1　先动手操作

与其啰啰嗦嗦地写上一大堆，还不如实际动手开发来得轻松，我们这就开始吧。而且我们一上来就完全抛开前面的说明，既不用C语言，也不用汇编程序，而是采用一个迥然不同的工具来进行开发（笑）。

■■■■■

有一种工具软件名为“二进制编辑器”（Binary Editor）[1]，是一种能够直接对二进制数进行编辑的软件。我们现在要用它来编辑出下图这样的文件。

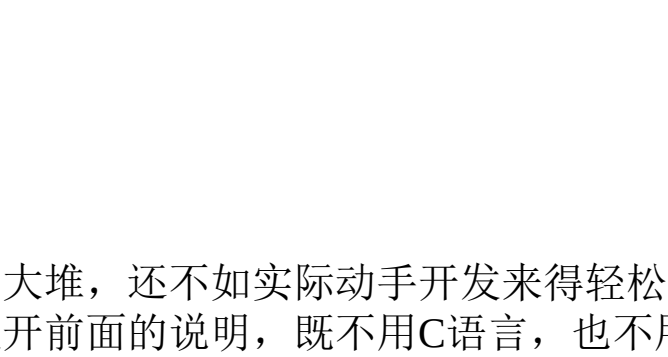
[1] 原文直译为“二进制编辑器”（Binary Editor），在中国“二进制编辑器”、“十六进制编辑器”这两种说法都有，这里尊重原著保留了“二进制编辑器”的说法。——译者注

也许有人会说“这样的工具我从来没有见过呀”，没关系，下面我们来详细地介绍一下。

首先打开下面这个网页：

http://www.vcraft.jp/soft/bz.html[2]

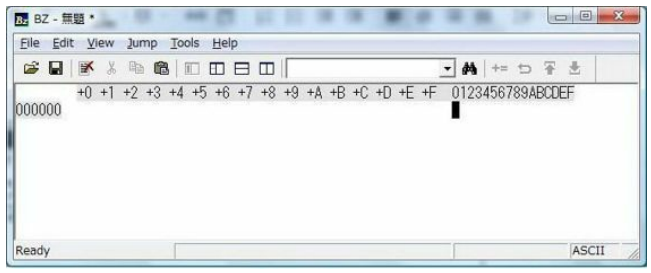
[2] 如果此网页连接不上，也可用google等检索工具来搜索一下，从别处下载Bz1621.lzh。



用**BZ**打开**helloos.img**时的画面

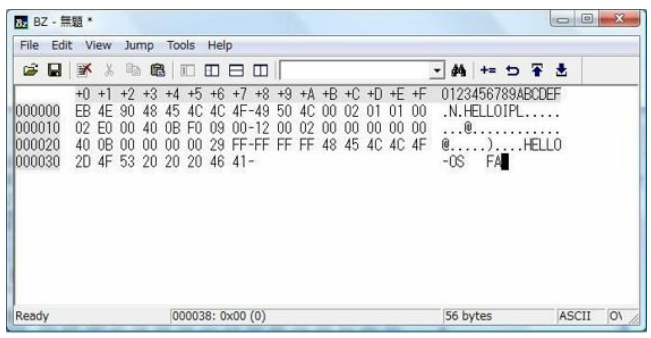
点击“在此下载”（Download）的链接，下载文件Bz1621.lzh　（在此非常感谢c.mos公司无偿公开这么好的软件）。当你读到本书的时候，也许会有新的版本发布，所以文件名可能会有所不同。接下来，安装下载下来的文件，然后双击启动Bz.exe程序。如果不能正常启动的话，可以参考上面网页的“★注意★”一项，按照上面的安装指导进行操作。

顺利启动的话屏幕上会出现如下画面。



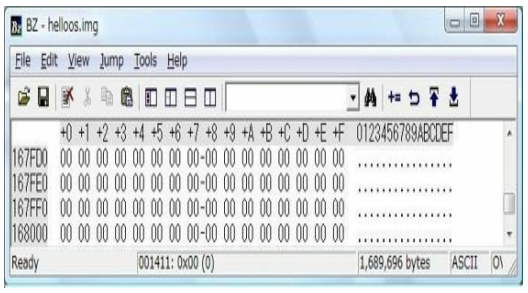
BZ起动时的画面

好，让我们赶紧来输入吧，只要从键盘上直接输入EB4E904845……就可以了，简单吧。其中字符之间的空格是这个软件在显示时为方便阅读自动插入的，不用自己从键盘上输入。另外，右边的.N.HELLOIPL……部分，也不用从键盘输入，这是软件自动显示的。可能版本或者显示模式不一样的时候，右侧显示的内容会与下面的截图有所不同。不过不用往心里去，这些内容完全是锦上添花的东西，即使不一样也没事。



输入到**000037**位置时的画面

从000090开始后面全都是00，一直输入到最后168000这个地址。如果一直按着键盘上的“0”不放手的话，画面上的0就会不停地增加，但因为个数相当多，也还是挺花时间的。如果家里有只猫的话，倒是可以考虑请它来帮忙按住这个键（日本的谚语：想让猫来搭把手，形容人手不足，连猫爪子都想借用一下），或者也可以干脆就用透明胶把这个键粘上。



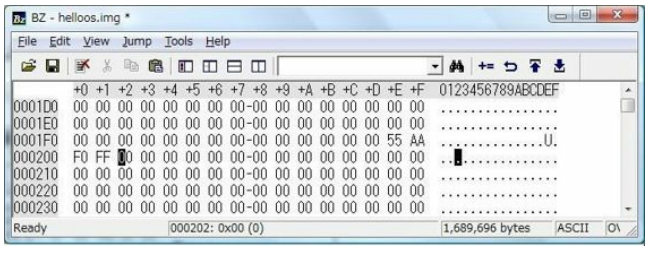
168000附近的画面

因为一下子输入到最后实在是挺花时间的，大家也许想保存一下中间结果，这时可以从菜单上选择“文件”（File）→“另存为”（Save As），画面上就会弹出保存文件

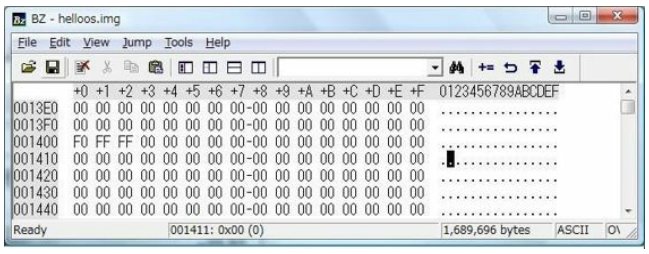
的对话框。我们可以随便取个名字进行保存，笔者推荐使用“helloos.img”。当想要打开保存过的文件时，首先要启动Bz.exe，从菜单上选择“文件”（File）→“打开”（Open），然后选择目标文件，这样原来保存的内容就能显示出来了。可是这个时候不管我们怎么努力按键盘，它都一点反应也没有。这是怎么回事？难道必须要一次性输入到最后吗？这个大家不必担心，其实只要从菜单里选择“编辑”（Edit）→“只读”（Read Only）就可以进入编辑状态啦。好了，我们继续输入。

如果家里的猫自由散漫惯了，不肯帮忙，而大家又不想用透明胶粘键盘这种土方法的话，不妨这样：用鼠标选择一部分0，然后从菜单选择“编辑”（Edit）→“复制”（Copy）。简简单单复制粘贴几次就可以大功告成了，这工具还真方便呀。

哦，对了，差点忘记一件重要的事——在地址0001F0和001400附近还有些地方不全是00，要像下图那样把它们也改过来，然后整体检查一下，确认没有输入错误。

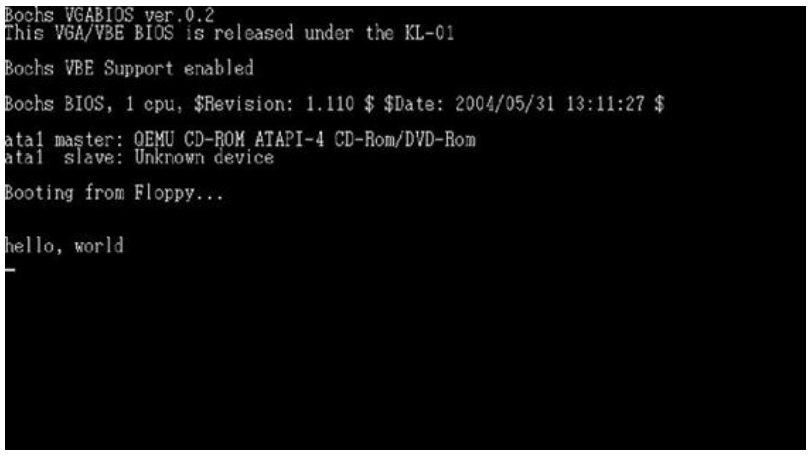


0001F0附近



001400附近

下面，我们把输入的内容保存下来就完成了软盘映像文件的制作，这时查看一下文件属性，应该能看到文件大小正好是1474560字节（=1440×1024字节）。然后我们将这个文件写入软盘（具体后述），并用它来启动电脑。如下所示，画面上会显示出“hello, world”这个字符串。目前的程序虽然简单，但毕竟一打开电脑它就能够自动启动，还能在屏幕上显示出一句话来，已经小小成功了哦。不过，我们现在还没有结束这个程序的方法，所以想要结束的时候，只能把软盘取出来后切断电脑电源，或者重新启动。



至于最关键的往磁盘上写映像文件的方法，笔者已经预先准备好了一个程序。在介绍它的使用方法之前，我们先把笔者准备的工具全都安装进来吧，这样后面讲解起来比较省事。下面我们就来看怎么安装这些工具。

■■■■■

打开附带光盘，里面有一个名为tolset[3]的文件夹，把这个文件夹复制到硬盘的任意一个位置上。现在里面的东西还不多，只有3MB左右，不过以后我们自己开发的软件也都要放到这个文件夹里，所以往后它会越来越大，因此硬盘上最好留出100MB左右的剩余空间。工具安装到此结束，我们既不用修改注册表，也不用设定路径参数，就这么简单。而且以后不管什么时候，都可以把这整个文件夹移动到任何其他地方。用这些工具，我们不仅可以开发操作系统，还可以开发简单的Windows应用程序或OSASK应用程序等。

[3] tool set的缩写，“工具套件”的意思。

接下来我们打开刚才安装的tolset文件夹，在文件夹的名字上单击鼠标右键，从弹出的菜单上选择“新建”（New）→“文件夹”（Folder）。画面上会显示出缺省的文件夹名“新建文件夹”（New Folder），我们要把它改为“helloos0”，并把前面保存的映像文件helloos.img复制到这个文件夹里。另外，刚才安装的tolset文件夹下有个名为z_new_w的子文件夹，其中有!cons_9x.bat和!cons_nt.bat这两个文件，要把它们也复制粘贴到helloos0文件夹里。

接着，在文件夹helloos0里单击鼠标右键，从弹出的菜单中选择“新建”（New）→“文本文件”（Text Document），并将文件命名为“run.bat”，回车后屏幕上会显示“如果改变文件扩展名，可能会导致文件不可用。确实要更改吗?”的对话框，我们选择“是”，创建run.bat文件。然后在run.bat文件名上单击鼠标右键，在弹出的菜单上选择“编辑”（Edit），输入下面内容并保存。

run.bat

```
copy helloos.img ..\z_tools\qemu\fdimage0.bin
..\z_tools\make.exe -C ../z_tools/qemu
```

然后按照同样的步骤，创建install.bat，并将下列内容输入进去。

install.bat

```
.. \z_tools\imgtol.com w a: helloos.img
```

其实以上步骤创建的所有文件都已经给事先给大家准备好了，就放在附带光盘中名为projects\01_day\helloos0的子文件夹里。所以大家只要把光盘上的helloos0复制下来，粘帖到硬盘的tolset文件夹里，所有的准备工作就瞬间完成了。

■■■■■

好了，现在我们就来把这个有点像操作系统的软件安装到软盘上吧。随便从附近的小店里买片新软盘来，在Windows下格式化一下（格式化方法：把软盘插入磁盘驱动器后打开“我的电脑”，在“3.5吋软盘”（3.5inches Floppy）A:上单击鼠标右键，再选择“格式化”（Format）即可）。对了，这个时候不要选择“快速格式化”选项。然后用鼠标左键双击helloos0文件夹里的 !cons_nt.bat文件（Windows95/98/Me的用户需要双击!cons_9x.bat），屏幕上就会出现一个命令行窗口（console）。我们先仔细确认一下软盘是否已经插好，然后在命令行窗口上输入“install”并回车，这样安装操作就开始了。稍候片刻，等安装程序执行完毕，我们的操作系统启动盘也就做好了。完成安装之后，也可以关闭刚才的命令行窗口了。

现在我们就用这张操作系统启动软盘来启动一下电脑试试吧，肯定跟刚才一样，会显示出“hello, world”的字样来。

在这里要提醒大家几点：一是软盘虽然不要求必须用全新的，但如果太旧的话，在读写过程中容易出问题，所以最好还是不要用太旧的软盘。另外，就算是新盘，如果太便宜的话有时也用不了，若是发现有问题，就需要再去买一张。最后一点，一旦格式化或者往软盘内安装操作系统，就会把里面原有的东西全部覆盖掉，所以大家千万不要用存有重要文件的软盘来尝试哦。

■■■■■

看到这里，大家可能会有各种问题：“这些我都明白，可是既要专门去买张软盘，又要重启电脑，实在太麻烦了，难道就没有什么更简单的方法吗？”、“我家的电脑根本就没有软驱呀”、“我的电脑没有什么重启按钮，也没有关电源的开关，一旦启动了这个奇怪的操作系统，就没法终止啦”。其实这些问题笔者已经考虑到了，所以特意准备了一个模拟器。我们有了这个模拟器，不用软盘，也不用终止Windows，就可以确认所开发的操作系统启动以后的动作，很方便呢。

使用模拟器的方法也非常简单，我们只需要在用!cons_nt.bat（或者是!cons_9x.bat）打开的命令行窗口中输入“run”指令就可以了。然后一个名叫QEMU的非常优秀的免费PC模拟器就会自动运行。QEMU不是笔者开发的，它是由国外的一些天才们开发出来的。感谢他们！

“我按照你说的一步一步地做了一遍，可是不行呀！怎么回事呢？”会遇到这种情况的人肯定是个非常认真的人，可能真的完全按照上面步骤用二进制编辑器自己做了一个helloos.img文件出来。出现这种问题，肯定是因为文件中有输入错误的地方，虽然笔者不知道具体错在哪儿，不过建议最好检查一下000000到000090，以及0001F0前后的数据。如果还是不行的话，那就干脆用附带光盘中笔者做的helloos.img好了。

可能有些人嫌麻烦，懒得自己输入，上来就直接使用光盘里的helloos.img文件，这当然也没什么不可以；但笔者认为这种体验（一点一点地输入，再千辛万苦地纠错，最终功夫不负有心人取得成功）本身更难能可贵，建议大家最好还是亲自尝试一下。

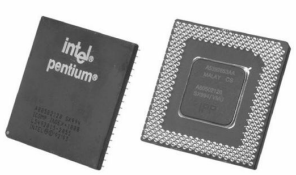
■■■■■

就这样，我们没有去改造现成的操作系统，而是从零开始开发了一个，并让它运转了起来（当然，如果别人承认这是个操作系统的话）。这太了不起了！大家完全可以在朋友们面前炫耀一番了。仅学习了几个小时开发的一个初学者，就能从零开始做出一个操作系统，这本书不错吧（笑）？这次我们考虑到从键盘直接输入比较麻烦，所以就只让它显示了一条消息；如果能再多输入一些内容的话，那仅用这种方法就可以开发任意一个操作系统（当然最大只能到1440KB）。现在唯一的问题是，我们还不知道之前输入的那些“EB 4E 90 48 45……”到底是什么意思（而这也正是我们所面临的最大问题）。今天剩下的时间，以及以后的29天时间里，我们都会讲解这个问题。

2　究竟做了些什么

为什么用这种方法就能开发出操作系统来呢？现在搞清楚这个问题，会对我们今后的理解很有帮助，所以在这里要稍做说明。

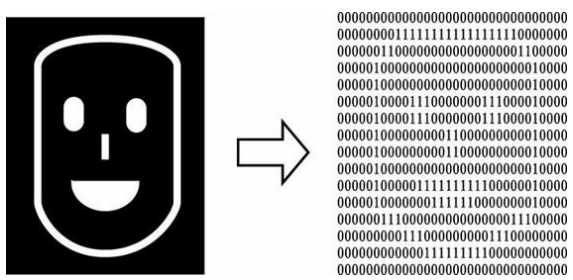
首先我们要了解电脑的结构。电脑的处理中心是CPU，即“central process unit”的缩写，翻译成中文就是“中央处理单元”，顾名思义，它就是处理中心。如果我们把别的元件当作中心来使用的话，那它就叫做CPU了，所以无论什么时候CPU都总是处理中心。不过这个CPU除了与别的电路进行电信号交换以外什么都不会，而且对于电信号，它也只能理解开（ON）和关（OFF）这两种状态，真是个没用的人呀（虽然它不是人吧，大家领会精神）。



CPU

我们平时会用电脑写文章、听音乐、修照片以及做其他各种各样的事情，我们用电脑所做的这些，其实本质上都不过是在与CPU交换电信号而已，而且电信号只有开（ON）和关（OFF）这两种状态。再说直白一点，CPU根本无法理解文章的内容，更不会鉴赏音乐、照片，它只会机械地进行电信号的转换。CPU有计算指令，所以它能够进行整数的加减乘除运算，也可以处理负数、计算小数以及10的100次方这样庞大的数值，它甚至能够处理我们初中才学到的平方根和高中才学到的对数、三角函数，而且所有这些计算仅通过一条指令就能简单实现。虽然CPU功能如此强大，但它其实根本不理解数的概念。CPU就是个集成电路板，它只是忠实地执行电信号给它的指令，输出相应的电信号。

这些概念可能不太容易理解，还是让我们来看个的具体例子吧。比如说，让我们用1来表示开（ON），用0来表示关（OFF），这样比较容易理解。我们可以用32×16=512个开（ON）和关（OFF）的集合（＝电信号的集合），来显示出下面这个不甚好看的人头像。



我们也可以用0000 0000 0000 0000 0000 0100 1010 0010 这32个电信号的集合来表示1186这个整数。（注：用二进制表示1186的话，就是100 1010 0010）。我们还可以用0100 1011 0100 1111 0100 1111 0100 0010这32个电信号的集合来表示“BOOK”这个单词（注：这实际上就是电脑内部保存这个单词时的电信号集合）。

CPU能看见的就只有这些开（ON）和关（OFF）的电信号。换句话说，假如我们给CPU发送这么一串电信号:

0000 0100 0011 1000 0000 1110 0001 0000

这信号可能是一幅画的部分数据，可能是个二进制整数，可能是一段音乐旋律，可能是文章中的一段文字，也可能是保存了的游戏的一部分数据，或者是程序中的一行代码，不管它是什么，CPU都一窍不通。CPU不懂这些，也不在乎这些，它只是默默地、任劳任怨地按照程序的指令进行相应的处理。

■■■■■

看到这里，或许有人会认为是先有了这么多要做的事情，所以人类才发明了CPU，而实际上并不是这样。最早人们发明CPU只是为了处理电信号，那个时候没有人能想到它后来会成为这么有用的机器。不过后来人们发现，一旦把电信号的开（ON）/关（OFF）与数字0和1对应起来，就能将二进制数转换为电信号，同时电信号也可以转换回二进制数。所以，虽然CPU依然只能处理电信号，但它从此摇身一变，成了神奇的二进制数计算机。

因为我们可以把十进制数转换成二进制数，也能把二进制数还原成十进制数，所以人们又发明了普通的计算机。后来，我们发现只要给每个文字都编上号（即文字编码），就可以建立一个文字与数字的对应关系，从而就可以把文字也转换成电信号，让CPU来处理文章（比如进行文字输入或者字词检索等）。依此类推，人们接着又找到了将图像、音乐等等转换成电信号的方法，使CPU的应用范围越来越广。不过CPU还是一如既往，只能处理电信号。

而且我们能用CPU来处理的并不仅仅只有数据，我们还可以用电信号向CPU发出指令。其实我们所编写的程序最终都要转换成所谓的机器语言，这些机器语言就是以电信号的形式发送给CPU的。这些机器语言不过就是一连串的指令代码，实际上也就是一串0和1的组合而已。

软盘的原理也有异曲同工之妙，简单说来，就是把二进制的0和1转换为磁极的N极和S极而已，所以我们只用0和1就可以写出映像文件来。不仅是映像文件，计算机所能处理的各种文件最终都是用0和1写成的。因此可以说，不能仅用0和1来表达的内容，都不能以电信号的形式传递给CPU，所以这种内容是计算机所无法处理的。

■■■■■

而“二进制编辑器”就是用来编辑二进制数的，我们可以很方便地用它来输入二进制数，并保存成文件。所以它就是我们的秘密武器，也就是说只要有了二进制编辑器，随便什么文件我们都能做出来。（厉害吧！）如果大家在商店里看到一个软件，很想要而又不想花那么多钱的话，那就干脆就回家用二进制编辑器自己做一个算啦！用这个方法我们完全可以自己制作出一个与店里商品一模一样的东西来。看上一个500万像素的数码相机，但是太贵了买不起？那有什么关系？我们只要有二进制编辑器在手，就可以制作出毫不逊色于相机拍摄效果的图像，而且想做几张就可以做几张。要是C编译器太贵了买不起，也不用郁闷。即使没有C编译器，我们也可以用二进制编辑器做出一个与编译器生成文件完全一样的执行文件，而且就连

C编译器本身都可以用二进制编辑器做出来。

有了这么强大的工具，制作操作系统就是小菜一碟。道理就是这么简单，所以我们这次不费吹灰之力就做了个操作系统出来也是理所当然的。或许有人会想“就为了讲这么个小事，有必要长篇大论写这么多吗？”其实不然，如果我们对CPU的基础有了彻底的理解，以后的内容就好懂多了。

■■■■■

“喂，且慢，我明白了二进制编辑器就是编辑二进制数的软件，可是在你让我输入你的helloos.img的时候，除了0和1以外，不是还让我输入了很多别的东西吗？你看，第一个不就是E吗？这哪里是什么二进制数？分明是个英文字母嘛！”……噢，不好意思，这说得一点错都没有。

虽然二进制数与电信号有很好的一一对应关系，但它有一个缺点，那就是位数实在太多了，举个例子来说，如果我们把1234写成二进制数，就成了10011010010，居然长达11位。而写成十进制数，只用4位就够了。因为这样也太浪费纸张了，所以计算机业界普遍使用十六进制数。十进制数的1234写成十六进制数，就是4D2，只用3位就够了。

那为什么非要用十六进制数呢，用十进制数不是也挺好的吗？实际上，我们可以非常简便地把二进制数写成十六进制数。
二进制数和十六进制数对照表

0000 – 0	0100 – 4	1000 – 8	1100 – C
0001 – 1	0101 – 5	1001 – 9	1101 – D
0010 – 2	0110 – 6	1010 – A	1110 – E
0011 – 3	0111 – 7	1011 – B	1111 – F

有了这个对照表，我们就能轻松进行二进制与十六进制之间的转换了。将二进制转换为十六进制时，只要从二进制数的最后一位开始，4位4位地替换过来就行了。
如：

100 1101 0010 → 4D2

反之，把十六进制数的4D2转换为二进制数的100 1101 0010也很简单，只要用上面的对照表反过来变换一下就行了。而十进制数变换起来就没这么简单了。同理，八进制数是把3位一组的二进制数作为一个八进制位来变换的，这种计数法在计算机业界也偶有使用。

因此我们在输入EB的时候，实际上是在输入11101011，所以它其实是个十六进制编辑器，但笔者习惯称它为二进制编辑器，希望大家不要见怪。

■■■■■

虽然笔者对二进制编辑器如此地赞不绝口，但用它也解决不了什么实际问题。因为这就相当于“只要有了笔和纸，什么优秀的小说都能写出来”一样。笔和纸不过就是笔和纸而已，实际上对创作优秀的小说也帮不上多大的忙。所以大家在写程序时，用的都是文本编辑器和编译器，没有谁只用二进制编辑器来做程序的。大家照相用的也都是数码照相机，没有谁只用二进制编辑器来做图像文件。因此，我们用二进

制编辑器进行的开发就到此为止，接下来我们要调转方向，开始用编程语言来继续我们的开发工作。不过有了这次的经验，我们就知道了如果今后遇到什么特殊情况还可以使用二进制编辑器，它是非常有用的。而且后面章节中我们偶尔也会用到它。

3　初次体验汇编程序

好，现在就让我们马上来写一个汇编程序，用它来生成一个跟刚才完全一样的helloos.img吧。我们这次使用的汇编语言编译器是笔者自己开发的，名为“nask”，其中的很多语法都模仿了自由软件里享有盛名的汇编器“NASM”，不过在“NASM”的基础之上又提高了自动优化能力。

超长的源代码

```
DB  0xeb, 0x4e, 0x90, 0x48, 0x45, 0x4c, 0x4c, 0x4f
DB  0x49, 0x50, 0x4c, 0x00, 0x02, 0x01, 0x01, 0x00
DB  0x02, 0xe0, 0x00, 0x40, 0x0b, 0xf0, 0x09, 0x00
DB  0x12, 0x00, 0x02, 0x00, 0x00, 0x00, 0x00, 0x00
DB  0x40, 0x0b, 0x00, 0x00, 0x00, 0x00, 0x29, 0xff
（为节省纸张，这里省略中间的18万4314行）
DB  0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00
```

我们使用复制粘帖的方法，就可以写出这样一个超长的源代码来，将其命名为“helloos.nas”，并保存在helloos0中。仔细看一下就能发现这个文件内容与我们用二进制编辑器输入的内容是一模一样的。

接着，我们用“!cons_nt.bat”或是“!cons_9x.bat”（我们在前面已经说过，要根据Windows的版本决定用哪一个。以后每次都这样解释一遍的话比较麻烦，所以我们就将它简写为!cons好了）　打开一个命令行窗口（console），输入以下指令（提示符部分不用输入）：

```
提示符[1]>..\z_tools\nask.exe helloos.nas helloos.img
```

[1] prompt，出现在命令行窗口中，提示用户进行输入的信息。

这样我们就得到了映像文件helloos.img。

好，我们的第一个汇编语言程序就这样做成了！……不过这么写程序也太麻烦了，要做个18万行的程序，不但浪费时间，还浪费硬盘空间。与其这样还不如用二进制编辑器呢，不用输入“0x”、“,”什么的，还能轻松一点。

■■■■■

其实要解决这个问题并不难，如果我们不只使用DB指令，而把RESB指令也用上的话，就可以一下将helloos.nas缩短了，而且还能保证输出的内容不变，具体我们来看下面。

正常长度的源程序

```
DB  0xeb, 0x4e, 0x90, 0x48, 0x45, 0x4c, 0x4c, 0x4f
DB  0x49, 0x50, 0x4c, 0x00, 0x02, 0x01, 0x01, 0x00
DB  0x02, 0xe0, 0x00, 0x40, 0x0b, 0xf0, 0x09, 0x00
DB  0x12, 0x00, 0x02, 0x00, 0x00, 0x00, 0x00, 0x00
DB  0x40, 0x0b, 0x00, 0x00, 0x00, 0x00, 0x29, 0xff
DB  0xff, 0xff, 0xff, 0x48, 0x45, 0x4c, 0x4c, 0x4f
```

```
DB  0x2d, 0x4f, 0x53, 0x20, 0x20, 0x20, 0x46, 0x41
DB  0x54, 0x31, 0x32, 0x20, 0x20, 0x20, 0x00, 0x00
RESB    16
DB  0xb8, 0x00, 0x00, 0x8e, 0xd0, 0xbc, 0x00, 0x7c
DB  0x8e, 0xd8, 0x8e, 0xc0, 0xbe, 0x74, 0x7c, 0x8a
DB  0x04, 0x83, 0xc6, 0x01, 0x3c, 0x00, 0x74, 0x09
DB  0xb4, 0x0e, 0xbb, 0x0f, 0x00, 0xcd, 0x10, 0xeb
DB  0xee, 0xf4, 0xeb, 0xfd, 0x0a, 0x0a, 0x68, 0x65
DB  0x6c, 0x6c, 0x6f, 0x2c, 0x20, 0x77, 0x6f, 0x72
DB  0x6c, 0x64, 0x0a, 0x00, 0x00, 0x00, 0x00, 0x00
RESB    368
DB  0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x55, 0xaa
DB  0xf0, 0xff, 0xff, 0x00, 0x00, 0x00, 0x00, 0x00
RESB    4600
DB  0xf0, 0xff, 0xff, 0x00, 0x00, 0x00, 0x00, 0x00
RESB    1469432
```

我们自己动手输入这段源程序比较麻烦，所以笔者把它放在附带光盘的projects\01_day\　helloos1目录下了。大家只要把helloos1文件夹复制粘帖到tolset文件夹里就可以了。之前的helloos0文件夹以后就不用了，我们可以把它删除，也可以放在那里留作纪念。顺便说一下，笔者将helloos0文件夹名改为了helloos1，删掉了其中没用的文件，新建并编辑了需要用到的文件，这样就做出了新的helloos1文件夹。操作系统就是这样一点一点地成长起来的。

每次进行汇编编译的时候，我们都要输入刚才的指令，这太麻烦了，所以笔者就做了一个批处理文件[2]asm.bat。有了这个批处理文件，我们只要在用“!cons”打开的命令行窗口里输入“asm”，就可以生成helloos.img文件。在用“asm”作成img文件后，再执行“run”指令，就可以得到与刚才一样的结果。

[2] batch file，基本上只是将命令行窗口里输入的命令写入文本文件。虽然还有功能更强的处理，但本书中我们用不到。所谓批处理就是批量处理，即一次处理一连串的命令。

■■■■■

DB指令是“define　byte”的缩写，也就是往文件里直接写入1个字节的指令。笔者喜欢用大写字母来写汇编指令，但小写的“db”也是一样的。

在汇编语言的世界里，这个指令是程序员的杀手锏，也就是说只要有了DB指令，我们就可以用它做出任何数据（甚至是程序）。所以可以说，没有用汇编语言做不出来的文件。文本文件也好，图像文件也好，只要能叫上名的文件，我们都能用汇编语言写出来。而其他的语言（比如C语言）就没有这么万能。

RESB指令是“reserve　byte”的略写，如果想要从现在的地址开始空出10个字节来，就可以写成RESB　10，意思是我们预约了这10个字节（大家可以想象成在对号入座的火车里，预订了10个连号座位的情形）。而且nask不仅仅是把指定的地址空出来，它还会在空出来的地址上自动填入0x00，所以我们这次用这个指令就可以输出很多的0x00，省得我们自己去写18万行程序了，真是帮了个大忙。

这里还要说一下，数字的前面加上0x，就成了十六进制数，不加0x，就是十进制数。这一点跟C语言是一样的。

4　加工润色

刚才我们把程序变成了短短的22行，这成果令人欣喜。不过还有一点不足就是很难看出这些程序是干什么的，所以我们下面就来稍微改写一下，让别人也能看懂。改写后的源文件增加到了48行，它位于附带光盘的projects\01_day\helloos2目录下，大家可以直接把helloos2文件夹复制到tolset里。现在helloos1也可以删掉了（每个文件夹都是独立的，用完之后就可以删除，以后不再赘述。当然放在那里留作纪念也是可以的）。

现在的程序有50行，也占不了多少地方，所以我们将它写在下面了。

有模有样的源代码

```
; hello-os
; TAB=4

; 以下这段是标准FAT12格式软盘专用的代码

        DB      0xeb, 0x4e, 0x90
        DB      "HELLOIPL"      ; 启动区的名称可以是任意的字符串（8字节）
        DW      512             ; 每个扇区（sector）的大小（必须为512字节）
        DB      1               ; 簇（cluster）的大小（必须为1个扇区）
        DW      1           ; FAT的起始位置（一般从第一个扇区开始）
        DB      2           ; FAT的个数（必须为2）
        DW      224         ; 根目录的大小（一般设成224项）
        DW      2880        ; 该磁盘的大小（必须是2880扇区）
        DB      0xf0        ; 磁盘的种类（必须是0xf0）
        DW      9           ; FAT的长度（必须是9扇区）
        DW      18          ; 1个磁道（track）有几个扇区（必须是18）
        DW      2           ; 磁头数（必须是2）
        DD      0           ; 不使用分区，必须是0
        DD      2880        ; 重写一次磁盘大小
        DB      0,0,0x29        ; 意义不明，固定
        DD      0xffffffff      ;（可能是）卷标号码
        DB      "HELLO-OS   "   ; 磁盘的名称（11字节）
        DB      "FAT12   "      ; 磁盘格式名称（8字节）
        RESB    18              ; 先空出18字节

; 程序主体
        DB      0xb8, 0x00, 0x00, 0x8e, 0xd0, 0xbc, 0x00, 0x7c
        DB      0x8e, 0xd8, 0x8e, 0xc0, 0xbe, 0x74, 0x7c, 0x8a
        DB      0x04, 0x83, 0xc6, 0x01, 0x3c, 0x00, 0x74, 0x09
        DB      0xb4, 0x0e, 0xbb, 0x0f, 0x00, 0xcd, 0x10, 0xeb
        DB      0xee, 0xf4, 0xeb, 0xfd

; 信息显示部分

        DB      0x0a, 0x0a      ; 2个换行
        DB      "hello, world"
        DB      0x0a            ; 换行
        DB      0

        RESB    0x1fe-$         ; 填写0x00,直到 0x001fe
        DB      0x55, 0xaa

; 以下是启动区以外部分的输出

        DB      0xf0, 0xff, 0xff, 0x00, 0x00, 0x00, 0x00, 0x00
        RESB    4600
        DB      0xf0, 0xff, 0xff, 0x00, 0x00, 0x00, 0x00, 0x00
        RESB    1469432
```

■■■■■

这里有几点新内容，我们逐一来看一下。首先是“;”命令，这是个注释命令，相当于C语言或是C++中的“//”。正是因为有它，我们才可以在源代码里加入很多注释。

其次是DB指令的新用法。我们居然可以直接用它写字符串。在写字符串的时候，汇编语言会自动地查找字符串中每一个字符所对应的编码，然后把它们一个字节一个字节地排列起来。这个功能非常方便，也就是说，当我们想要变更输出信息的时候，就再也不用自己去查字符编码表了。

再有就是DW指令和DD指令，它们分别是“define word”和“ddefine double-word”的缩写，是DB指令的“堂兄弟”。word的本意是“单词”，但在计算机汇编语言的世界里，word指的是“16位”的意思，也就是2个字节。“double-word”是“32位”的意思，也就是4个字节。

对了，差点忘记说RESB 0x1fe-$了。这个美元符号的意思如果不讲，恐怕谁也搞不明白，它是一个变量，可以告诉我们这一行现在的字节数（如果严格来说，有时候它还会有别的意思，关于这一点我们明天再讲）。在这个程序里，我们已经在前面输出了132字节，所以这里的$就是132。因此nask先用0x1fe减去132，得出378这一结果，然后连续输出378个字节的0x00。

那这里我们为什么不直接写378，而非要用$呢？这是因为如果将显示信息从“hello, world”变成“this is a pen.”的话，中间要输出0x00的字节数也会随之变化。换句话说，我们必须保证软盘的第510字节（即第0x1fe字节）开始的地方是55 AA。如果在程序里使用美元符号（$）的话，汇编语言会自动计算需要输出多少个00，我们也就可以很轻松地改写输出信息了。

■■■■■

既然可以毫不费力地改写显示的信息，就一定要好好发挥这一功能，让我们的操作系统显示出自己喜欢的一句话，让它成为一个只属于我们自己的、世界上独一无二的操作系统。不过遗憾的是现在它还不能显示汉字。当然大家也可以尝试一下，但由于这个程序还没有显示汉字的功能，所以显示出来的都是乱码，因此大家先将就一下，用英语或拼音吧。

■■■■■

最后再给大家解释一下程序中出现的几个专门术语。时间不早了，我们今天就到这吧。其他的留待明天再说。

TAB=4 ……	有的文本编辑器可以调整TAB键的宽度。请使用这种编辑器的人将TAB键的宽度设定成4，这样源程序更容易读。可能有人说，我这里只能用记事本（notepad），TAB键宽度固定为8，想调都没法调。没关系，明天笔者来推荐一个好用的文本编辑器。
FAT12格式 …	（FAT12 Format）用Windows或MS-DOS格式化出来的软盘就是这种格式。我们的helloos也采用了这种格式，其中容纳了我们开发的操作系统。这个格式兼容性好，在Windows上也能用，而且剩余的磁盘空间还可以用来保存自己喜欢的文件。
启动区 ……	（boot sector）软盘第一个的扇区称为启动区。那么什么是扇区呢？计算机读写软盘的时候，并不是一个字节一个字节地读写的，而是以512字节为一个单位进行读写。因此,软盘的512字节就称为一个扇区。一张软盘的空间共有1440KB，也就是1474560字节，除以512得2880，这也就是说一张软盘共有2880个扇区。那为什么第一个扇区称为启动区呢？那是因为计算机首先从最初一个扇区开始读软盘，然后去检查这个扇区最后2个字节的内容。 如果这最后2个字节不是0x55 AA，计算机会认为这张盘上没有所需的启动程序，就会报一个不能启动的错误。（也许有人会问为什么一定是0x55 AA呢？那是当初的设计者随便定的，笔者也没法解释）。如果计算机确认了第一个扇区的最后两个字节正好是0x55 AA，那它就认为这个扇区的开头是启动程序，并开始执行这个程序。
IPL ……	initial program loader的缩写。启动程序加载器。启动区只有区区512字节，实际的操作系统不像hello-os这么小，根本装不进去。所以几乎所有的操作系统，都是把加载操作系统本身的程序放在启动区里的。有鉴于此，有时也将启动区称为IPL。但hello-os没有加载程序的功能，所以HELLOIPL这个名字不太顺理成章。如果有人正义感特别强，觉得“这是撒谎造假，万万不能容忍！”，那也可以改成其他的名字。但是必须起一个8字节的名字，如果名字长度不到8字节的话，需要在最后补上空格。
启动 ……	（boot）boot这个词本是长靴（boots）的单数形式。它与计算机的启动有什么关系呢？一般应该将启动称为start的。实际上，boot这个词是bootstrap的缩写，原指靴子上附带的便于拿取的靴带。但自从有了《吹牛大王历险记》（德国）这个故事以后，bootstrap这个词就有了“自力更生完成任务”这种意思（大家如果对详情感兴趣，可以在Google上查找，也可以在帮助和支持网页http://hrb.osask.jp 上提问）。而且，磁盘上明明装有操作系统，还要说读入操作系统的程序（即IPL）也放在磁盘里，这就像打开宝物箱的钥匙就在宝物箱里一样，是一种矛盾的说法。这种矛盾的操作系统自动启动机制，被称为bootstrap方式。boot这个说法就来源于此。如果是笔者来命名的话，肯定不会用bootstrap 这么奇怪的名字，笔者大概会

叫它“多级火箭式”吧。