**Hex 文件格式**



:04 cddc 00 ac0e0020 77

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 长度 | 地址 | 类型 | 数据 | 校验 |
| :04 | cddc | 00 | Ac0e0020 | 77 |
| 4Byte | 0xcddc 高前低后 | 数据 |  |  |

第一个字节:表示本行的数据长度.

第二个,第三个字节表示本行数据的起始地址.

第四字节表示数据类型，数据类型有：0x00、0x01、0x02、0x03、0x04、0x05。

'00' Data Rrecord：用来记录数据，HEX文件的大部分记录都是数据记录

'01' End of File Record:用来标识文件结束，放在文件的最后，标识HEX文件的结尾

'02' Extended Segment Address Record:用来标识扩展段地址的记录

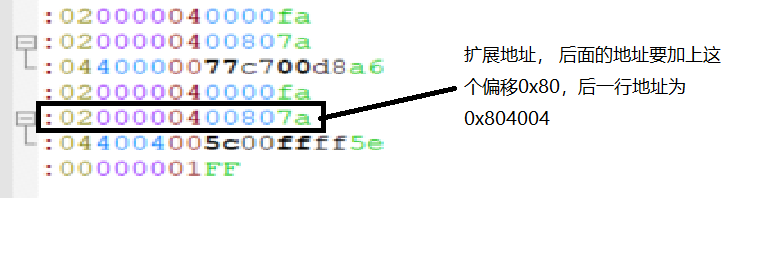
'03' Start Segment Address Record:开始段地址记录

'04' Extended Linear Address Record:用来标识扩展线性地址的记录

'05' Start Linear Address Record:开始线性地址记录

然后是数据，最后一个字节 为校验和。

校验和的算法为：计算校验和前所有16进制码的累加和(不计进位)，检验和 = 0x100 - 累加和.



<https://wenku.baidu.com/view/e2e23ebb4a73f242336c1eb91a37f111f0850d63.html>

以下信息来自：https://www.21ic.com/article/807068.html

**文件格式**

通过上面的文件，我们不难发现，hex文件每行都由:作为起始码，这是显而易见的，而后面这些乱七八糟的数据又代表什么呢？先看下面这张图；

Chart, treemap chart

Description automatically generatedhexformat

这个和一般的通讯协议类似，一帧数据往往包括起始码，数据长度，数据类型，数据，校验码等等，所以hex文件也不例外，这里它包含几个特点，下面参考了wiki；

* 起始码：每行数据作为一帧，并由 :作为起始码；
* 字节长度：两个十六进制数字（一对十六进制数字），指示数据字段中的字节数（十六进制数字对）。最大字节数为255（0xFF）。 16（0x10）和 32（0x20）是常用的字节数；
* 地址：四个十六进制数字，代表数据的16位起始存储器地址偏移量。数据的物理地址是通过将此偏移量添加到先前建立的基地址来计算的，因此允许内存寻址超过16位地址的64 KB限制。基地址默认为零，可以通过各种类型的记录进行更改。基地址和地址偏移量始终表示为大端值。
* 指令类型：两个十六进制数字00到05，定义了这行数据的具体含义；
* 数据：n字节数据序列，由2个n十六进制数字表示；
* 校验码：（两个十六进制数字），可以用来验证记录没有错误的计算值；

## 指令类型（Record type）

Record type的值一般是00~05，这表示了，当前这行hex格式的数据，所代表的含义：

| **Hex code** | **Record type** | **Description** | **Example** |
| --- | --- | --- | --- |
| **00** | 数据 | 包含数据和该数据的16位起始地址。字节计数指定记录中的数据字节数。右侧显示的示例为0B （十一）个数据字节（61， 64， 64， 72， 65， 73， 73， 20， 67， 61， 70）位于以地址开头的连续地址 0010。 | :0B0010006164647265737320676170A7 |
| **01** | 文件结束 | 每个文件在文件的最后一行必须恰好发生一次。数据字段为空（因此字节数为00），并且地址字段通常为 0000。 | :00000001FF |
| **02** | 扩展段地址 | 数据字段包含一个16位的段基址（因此字节数始终为02）与80x86实模式寻址兼容。地址字段（通常为0000）被忽略。最近的段地址02记录乘以16，然后加到每个后续数据记录地址，以形成数据的物理起始地址。这允许寻址多达1 MB的地址空间。 | :020000021200EA |
| **03** | 起始段地址 | 对于80x86处理器，请指定CS：IP寄存器的初始内容（即起始执行地址）。地址字段是0000，字节数始终为04，前两个数据字节是CS值，后两个是IP值。 | :0400000300003800C1 |
| **04** | 扩展线性地址 | 允许32位寻址（最大4GiB）。记录的地址字段将被忽略（通常是0000），其字节数始终为02。两个数据字节（大字节序）为所有后续类型指定32位绝对地址的高16位00记录; 这些高位地址位适用于下一个04记录。类型的绝对地址00 通过组合最近的高16位地址位形成记录 04 用低16位的地址记录 00记录。如果是类型00 记录之前没有任何类型 04 记录，然后其高16位地址位默认为0000。 | :02000004FFFFFC |
| **05** | 起始线性地址 | 地址字段是 0000（未使用），字节数始终为04。四个数据字节代表一个32位地址值（big-endian）。对于80386和更高版本的CPU，此地址将加载到EIP寄存器中。 | :0400000508000135B9 |

## 校验和

最后一个字节表示除了起始码之后，其余字节的校验和，

### :04 02B0 00 92020008 AE

0x04 + 0x02 + 0xB0 + 0x92 + 0x02 + 0x08 = 0x152 ~0x52 + 1 = 0xAE

### :04 0000 05 08000135 B9

0x04 + 0x05 + 0x80 + 0x01 + 0x35 = 0X47 ~0x47 + 1 = 0XB9

### :00 0000 01 FF

~0x01 + 1 = 0xFF

## hex转bin文件

资深调包侠找到一个可以使用的包；

pip install intelhex

Per example, converting content of foo.hex to foo.bin addresses **from** 0 to FF:  
  
$ python hex2bin.py -r 0000:00FF foo.hex  
Or (equivalent):  
  
$ python hex2bin.py -r 0000: -s 256 foo.hex

坐电梯到GitHub

https://github.com/python-intelhex/intelhex