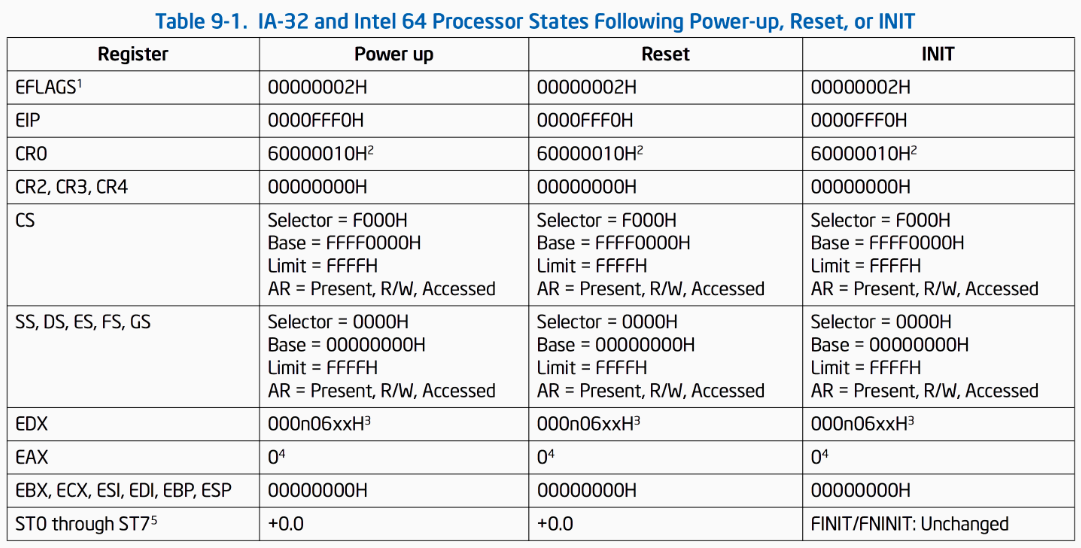
这个系列的首篇为大家介绍一下计算机是如何启动的。我的设想是把我们写出来的程序直接放在CPU上去跑，因为汇编语言就是和硬件打交道的语言，如果在我们的程序和硬件之间隔一层操作系统的话，总感觉没那么舒服。结合这篇文章的知识，我们就可以让硬件直接加载我们自己的程序并执行。

通常计算机的启动方式分为两种：传统的BIOS-MBR启动模式和新的UEFI-GPT启动模式。在这里我们介绍传统的BIOS-MBR启动模式。

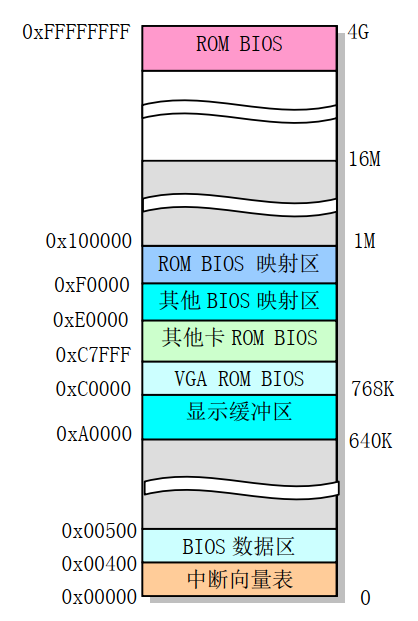
先来看一下按下计算机的电源或者复位键之后CPU中寄存器的初始值。

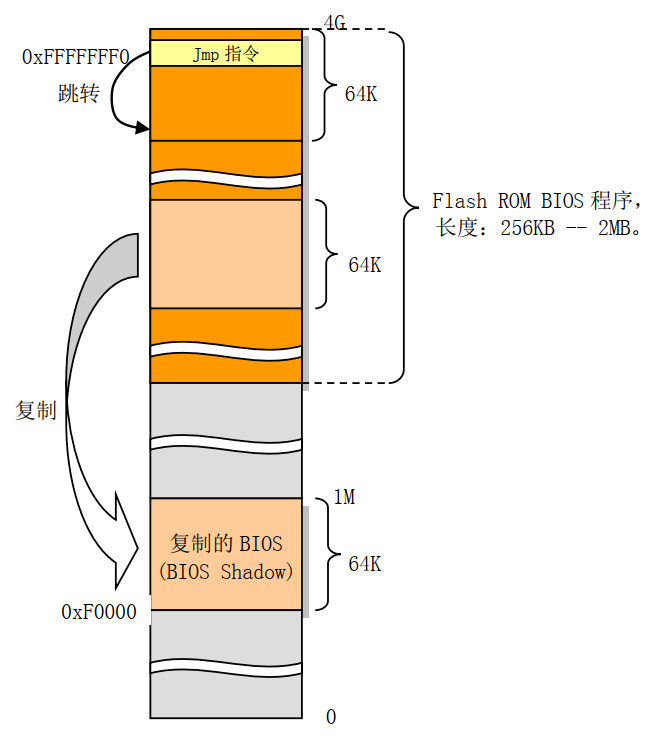


我们重点关注一下cs寄存器和eip寄存器，这两个寄存器决定了CPU执行哪里的代码，地址计算的方式为：EA = Base + EIP。

由上图的寄存器状态可知CPU在加电后首先会去0xFFFF0000 + 0x0000FFF0处即0xFFFFFFF0处执行第一条指令。下面我们看一看计算机加电后的内存状态是怎样的。

当计算机上电初始化时，物理内存被设置成从地址0开始的连续区域。除了地址从0xA0000到0xFFFFF(640K到1M共384K)和0xFFFE0000到0xFFFFFFFF(4G处的最后64K)范围以外的所有内存都可用作系统内存。这两个特定范围被用于I/O设备和BIOS程序。640K–1M之间的384K用作下图中指明的用途。其中地址0xA0000开始的128K用作显存缓冲区，随后部分用于其他控制卡的ROM BIOS或其映射区域，而0xF0000到1M范围用于高端系统ROM BIOS的映射区。



ROM-BIOS是一段固化在主板上的程序，这段程序在计算机加电后会自动被加载到内存中，主要用于计算机的自检和初始化。根据上面的分析可知0xFFFFFFF0正好处于这段程序中，位于4G空间最后一个64K的最后16字节处。这里会被安排一条jmp指令，用于跳转到BIOS代码中64KB范围内的某一条指令开始执行。BIOS在执行了一系列硬件检测和初始化操作之后，会把与原来PC机兼容的64KB BIOS代码和数据复制到内存低端1M末端的64K处，然后跳转到这个地方并让CPU运行在实地址模式下。过程如下图所示。

最后，如果硬盘或软盘是首选的启动设备的话，BIOS会读取其中的0柱面0磁道1扇区，并检测是否为可引导设备，如果是的话，这个扇区将被加载到内存0x7c00处并被执行。可引导的标志是扇区的最后两个字节为0x55和0xAA。

在这篇文章中我们先不写程序，但我们可以做一个小实验验证一下上面学到的这些知识。

实验环境如下：

* 系统：Ubuntu 18.04.4 LTS
* 虚拟机：QEMU emulator version 2.11.1(Debian 1:2.11+dfsg-1ubuntu7.21)
* 二进制编辑器：hexedit

我们将用虚拟机代替物理机，用磁盘映像文件代替真实的硬盘。因为实验中我们需要操作主引导记录，主引导记录对于操作系统是非常重要的，如果操作不慎将会导操作系统无法启动。

首先我们创建一个空的磁盘映像文件，使用dd命令。

|  |  |
| --- | --- |
| 1 | $ dd if=/dev/zero of=disk.img bs=1024 count=100 |

上面我们创建了一个100KB大的硬盘映像，好像有点小，但是对于我们的实验来说足够了。

查看一下硬盘映像中的内容。因为我们在创建时输入使用的是产生0的设备文件，所以现在的这块"硬盘"中的内容全部为零，为了加深印象我们还是查看一下。

|  |  |
| --- | --- |
| 1  2  3  4 | $ xxd -a disk.img  00000000: 0000 0000 0000 0000 0000 0000 0000 0000 ................  \*  00018ff0: 0000 0000 0000 0000 0000 0000 0000 0000 ................ |

可以看到disk.img中的内容全为零。

下面我们试试看如果直接用虚拟机去启动这块硬盘的话会发生什么。

|  |  |
| --- | --- |
| 1 | $ qemu-system-i386 disk.img |

虚拟机启动后结果如下，Boot failed: not a bootable disk。提示磁盘不可引导。

现在我们将disk.img文件的第510、511字节改为0x55、0xAA，然后重新启动看看结果。

关于如何直接修改这两个字节，如果有二进制文件编辑器的话会简单很多。当然如果没有的话我们可以自己写一个工具来完成这个目标。

先使用工具来直接对这两个字节进行编辑，我们用到的工具是hexedit。

|  |  |
| --- | --- |
| 1 | $ hexedit disk.img |

通过键盘方向键定位到位置0x1FE即十进制510这个位置，将连续的两个字节分别修改为0x55、0xAA，Ctrl + X保存退出。

再次查看，可以看出已经有了我们需要的可引导标记。

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7 | $ xxd -a disk.img  00000000: 0000 0000 0000 0000 0000 0000 0000 0000 ................  \*  000001f0: 0000 0000 0000 0000 0000 0000 0000 55aa ..............U.  00000200: 0000 0000 0000 0000 0000 0000 0000 0000 ................  \*  00018ff0: 0000 0000 0000 0000 0000 0000 0000 0000 ................ |

我们再次尝试启动虚拟机，看看这次的结果。

|  |  |
| --- | --- |
| 1 | $ qemu-system-i386 disk.img |

如我们所愿，虚拟机这次告诉我们已经从硬盘开始引导了。但是我们的硬盘里一行指令也没有，所以现在虚拟机就傻傻的在哪里等着。

至此我们已经知道了计算机是如何启动的，剩下的内容作为赠送给大家的一点小礼物。教大家写一个小工具，用于将主引导记录，也就是磁盘的0柱面0磁道1扇区，对应磁盘映像文件的前512个字节设置为可引导的。

我们的程序功能很简单：

1. 读入一个不大于510字节的文件
2. 将它补齐到510字节
3. 将第510、511字节(从0开始计数)设置为0x55、0xAA
4. 写入原文件

程序代码如下：

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39  40  41  42  43  44  45  46  47  48  49  50  51  52  53  54  55  56  57  58  59  60  61  62 | #include <fcntl.h>  #include <stdarg.h>  #include <stdio.h>  #include <stdlib.h>  #include <unistd.h>  static void die(const char \*str, ...)  {  va\_list args;  va\_start(args, str);  vfprintf(stderr, str, args);  va\_end(args);  fputc('\n', stderr);  exit(1);  }  int main(int argc, char \*argv[])  {  int fd;  ssize\_t size;  char buf[512];  if (argc != 2)  {  die("usage: ./sign bin\_file");  }  if ((fd = open(argv[1], O\_RDWR)) == -1)  {  die("open %s error", argv[1]);  }  if ((size = read(fd, buf, 1024)) == -1)  {  die("read %s error", argv[1]);  }  if (size > 510)  {  die("boot block too large: %d bytes (max 510)", size);  }  while ((510 - size) != 0)  {  buf[size++] = 0;  }  buf[510] = 0x55;  buf[511] = 0xAA;  if (lseek(fd, 0, SEEK\_SET) == -1)  {  die("lseek error");  }  if ((size = write(fd, buf, 512)) != 512)  {  die("write %s error", argv[1]);  }  return 0;  } |

程序的结构很简单，不多解释了。主逻辑之外封装了一个用于打印错误信息并退出的函数die，这个函数来自linux的源码，功能上没有什么深奥的地方，重点在于它接收可变参数。以后有时间了跟大家介绍一下C语言中可变参数怎么用，下面我们看一下这个工具怎么使用。

将文件保存为sign.c，编译链接：

|  |  |
| --- | --- |
| 1  2 | $ make sign  cc sign.c -o sign |

我们使用make来进行编译链接，可以少敲几个字母。真正执行的命令也在控制台打印出来了。

创建一个空文件，并查看：

|  |  |
| --- | --- |
| 1  2  3  4  5  6 | $ touch boot  $ ls -l  总用量 20  -rw-rw-r-- 1 laoli laoli 0 3月 4 14:15 boot  -rwxrwxr-x 1 laoli laoli 12776 3月 4 14:14 sign  -rw-rw-r-- 1 laoli laoli 928 3月 4 14:14 sign.c |

可以看到boot文件长度为0，内容为空。接下来用我们的工具处理一下这个文件：

|  |  |
| --- | --- |
| 1  2  3  4  5  6 | $ ./sign boot  $ ls -l  总用量 24  -rw-rw-r-- 1 laoli laoli 512 3月 4 14:16 boot  -rwxrwxr-x 1 laoli laoli 12776 3月 4 14:14 sign  -rw-rw-r-- 1 laoli laoli 928 3月 4 14:14 sign.c |

此时boot文件的长度已经是512字节了。查看其内容：

|  |  |
| --- | --- |
| 1  2  3  4 | $ xxd -a boot  00000000: 0000 0000 0000 0000 0000 0000 0000 0000 ................  \*  000001f0: 0000 0000 0000 0000 0000 0000 0000 55aa ..............U. |

用虚拟机从这个磁盘映像文件启动：

|  |  |
| --- | --- |
| 1 | $ qemu-system-i386 boot |

结果和之前使用hexedit手动编辑是一样的，引导成功。

之所以要教大家自己写一个创建可引导磁盘映像的工具是因为以后我们写的汇编代码需要用到。这篇文章中我们并没有涉及到汇编语言，所以虚拟机启动起来之后我们无法观察到太多的信息。下一篇文章中，老李将正式开始教大家学习汇编语言。加油，奥里给！