这是一篇工程向的文档，描述的内容如下：

我收集到了x86和64的指令集及其相关描述，在backend的xlsx文件里面。然而，首先我需要根据这个xlsx文件来获取C能够操作的数据。

其次，这个xlsx文档中，只是x86相关指令集所拥有的这些属性，这并不表示其他的指令集也同样有所有这些属性，而且也有几个字段，是这个xlsx文件参考的网站所自己弄的标签，比如考虑到后期的迁移，因此需要在读取了这些属性之后，仍然需要将其转化为通用的后端指令结构。

因此这篇文档描述了，1，将xlsx文件里面的属性读取并输出到某个H文件中，作为参考的x86后端指令集结果，2，将上一步的结果输出到”通用”的后端指令集的结构体中。

基于整个开发周期而言，还是用python来解析并输出个C文件吧，反正最后这个输出结果是作为mcc的一部分进行编译的，换言之，这个xlsx和py文件都不要，也不是什么大事情。至于输出的结果，我不想在这边进行更加深入的解析，主要原因是，这会使得后端格式更加复杂，所以全都是字符串就好了。

Python会生成一个ins\_raw的c文件，里面只有给定的字符串组成的结构体列表。

随后这个c文件被读取进来，然后进行进一步的处理，输入到x86\_ins中去。在这个过程中，会解析操作数等等东西，并判断其合法性（因为我是手工处理的xlsx，那个网页抓下来的时候，基本不整齐，数量还特别多，所以需要进行一些判断。）

除此之外，需要调用系统调用来判断CPU所支持的指令集，比如vmx sse这些指令集。如果没有，那么需要在待选指令集中去掉它——当然不是真的去掉，而是打个标记，万一生成的时候不支持就不生成了呗。但是比如内联汇编，总不能直接跟人家说生成不了吧。只报个warning就好了。

就目前来看，只有三个部分是比较重要的

指令名

操作数

Flag

描述

正如我找到这个的网页描述，操作数分为两部分，寻址方式和指令类型。两个拼起来，就得到了在指令集中出现的操作数类型。

Flag则分为两种，odiszapc和f87 FPU的1234格式。