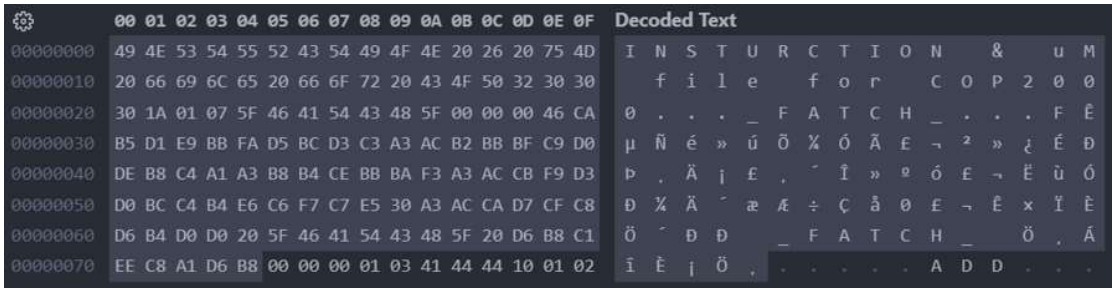
本文档使用原始的 INS 指令集作为演示，您可以导出 COP2000 的原始指令集或者直接运行一次 ins2table.py 文件即可生成原始指令集。
接下来对 INS 文件结构进行讲解：

下图是 INS 文件的首部 header，请不要修改。

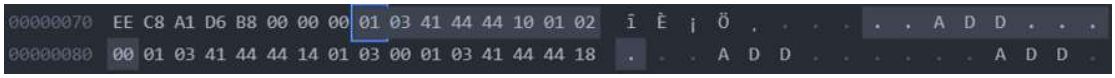


接下来，0x00 开头表示是一条空指令，共三条。

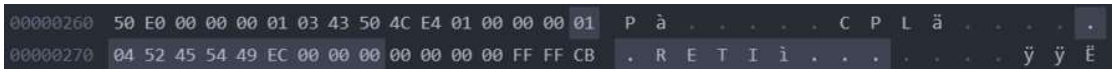
我们看一下 INS table 文件：

	A	B	C	D	E	F	G
1		mnemonic	microinstruction				address
2	1	_FATCH_	CBFFFF00	FFFFFF00	FFFFFF00	FFFFFF00	0x00
3	2						0x04
4	3						0x08
5	4						0x0C
6	5	ADD A,R?	FFF7EFFF	FFFE90FF	CBFFFFFF	FFFFFF00	0x10
7	6	ADD A,@I	FF77FFFF	D7BFEFFF	FFFE90FF	CBFFFFFF	0x14

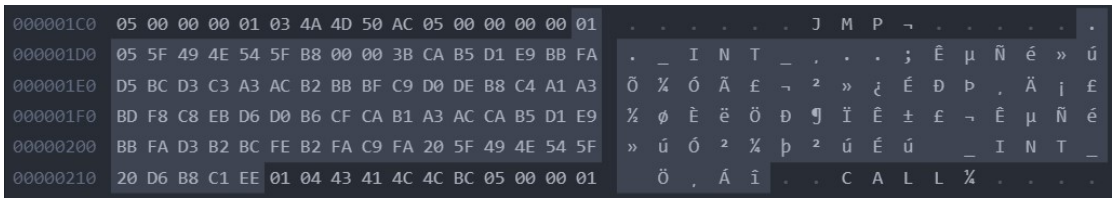
0x01 开头代表此处有指令，0x03 代表指令名占接下来的 3 个 byte.指令名之后的 0x10 标明了指令地址，0x01 代表第一个操作数是 A 寄存器，0x02 代表第二个操作数是 R?，0x00 是指令结束标志



最后一条 RETI 指令也一样。0x01 表示有指令，04 表示指令名占接下来的 4 个 byte.指令名之后的 0xEC 标明了指令地址，0x00 代表第一个操作数空，0x00 代表第二个操作数空，0x00 是指令结束标志。后面四条 0x00 标明接下来有四条空指令。



_INT_指令结构特殊，内容和位置都不允许修改。



指令描述结束后，是文件的微指令存储区。每条微指令有 4 个 byte，每条指令有四条微指令，共 64 条指令、256 条微指令。在 INS 文件中，每条微指令都需要 8 位 hex 表示。最后两位要么是 0x00，要么是 0xff。**正常来说，每条微指令的最后两位都是以 0xff 结尾，空微指令 0xffffff 才以 0x00 结尾。**不过也有一些例外，比如 READ、WRITE、_INT_等指令的微指令可能会以 0x00 结尾。具体原因未知，不过不建议改动这几条特殊指令。**微指令按字节逆序存储并补全末尾一字节，例如一条微指令 0xD7 0xEF 0xBF 会被存储为 0xBF 0xEF 0xD7 0xFF。**