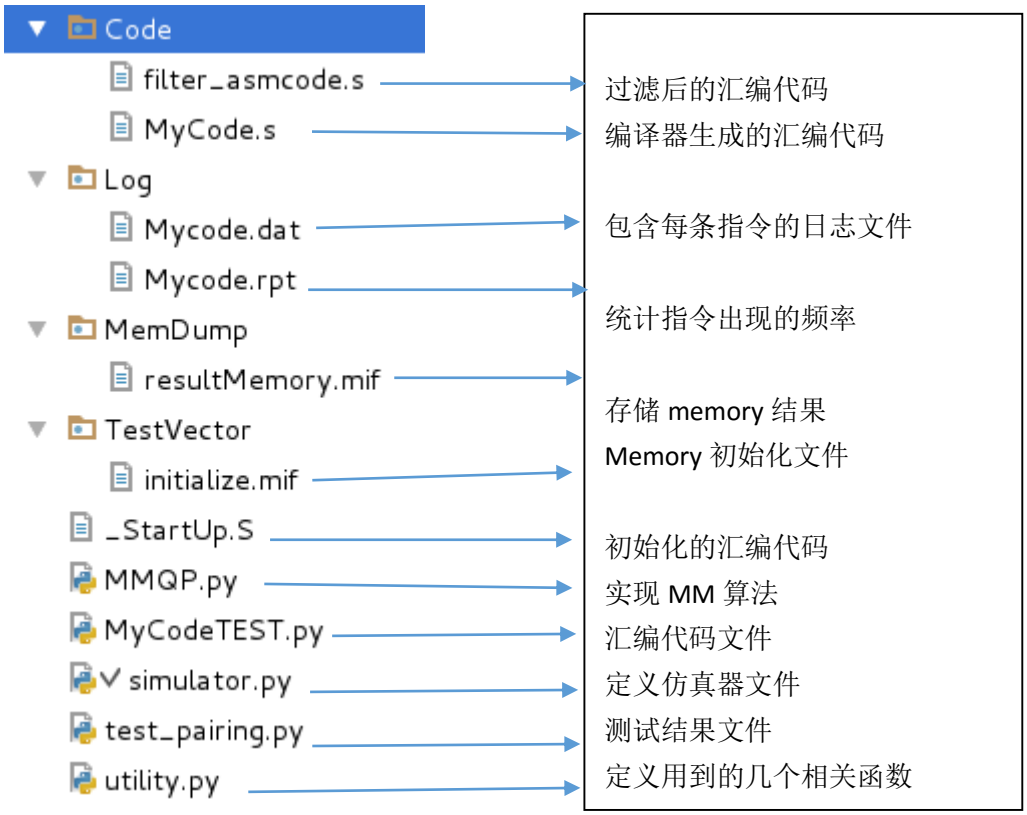
使用 python 语言实现一个能够测试汇编代码的仿真器
工程文件的说明如下：



首先 crypto 编译器输出的汇编代码有很多冗余信息，该信息对仿真功能没有影响，所以需要先过滤该信息得到纯粹的汇编代码，这个用 utility.py 中的 filter_asmCode()函数实现。

第二步，定义初始化 memory 文件，在/TestVector/initialize.mi 文件中 WIDTH 代表 memory 位宽，DEPTH 代表 memory 深度，也就是地址区间大小。下面索引号代表地址，之后的值代表改地址存储的值。

第三步，定义寄存器、每条汇编代码的语义，主要利用 python 中的 re 模块，进行正则表达式的匹配。在 simulator.py 中实现。

第四步，在_StartUp.S 中初始化一小段汇编代码，包括栈的分配，和初始化读入 memory 中的数据。

第五步，在 MyCodeTEST.py 中配置仿真条件，指定需要仿真的汇编代码和开始仿真的标号。

第六步，从 MemDump/resultMemory 中观察 memory 输出结果。

测试：

以 opt pairing 程序为例进行测试。

test_pairing.py 中定义了 pairing 的实现，已知功能是正确的。汇编代码中 Paring 的运算都需要在蒙哥马利域上计算，所以汇编程序的数据输入和输出都是蒙哥马利域上的数，结果需要转化为素域上的数，再和正确的 pairing 结果作比较。

Memory 存储格式说明：

st64，ld64,指令虽然本意是 64 位操作数 store 和 load 指令，但是在这里看作是 320 位宽的操作数指令，（因为该 pairing 的计算结果大概在 280 多位宽，为了 4 字节对齐，所以

swr 和 gwr 寄存器的位宽定为 320bits，所以这个指令在这里只是象征性的意义，在硬件上是要用 320bits 实现的），而存储器的位宽设置为 160bits，所以一个 st64 和 ld64 需要操作相邻的两个地址，低地址存放高 160bits，低地址存放低 160bits，所以每个 st32 和 ld32 指令也都把 32bits 的操作数存放在 160bits 中。

上述方法针对 320bits 的操作数 store 和 load 是有利的，但是针对 32bits 的操作数要存放在 160bits 地址中，造成了硬件上的浪费。Pairing 计算一次，ld64 指令出现 317747 次，st64 指令出现 229357 次，st32 指令出现 276038 次,ld32 指令出现 474028 次，st32 和 ld32 出现的频率较高，所以按照上述方法的话会极大造成硬件的浪费。尝试的解决办法是:如果编译器可以生成 st320 和 ld320 指令，那么就可以把存储器数据位宽设为 32bits，而 320bits 操作数的存取可以采用向量指令的方法，(因为存放地址是连续的)，该办法 320bits 操作数的存取性能不会差,32bits 操作数也没有造成硬件的浪费。