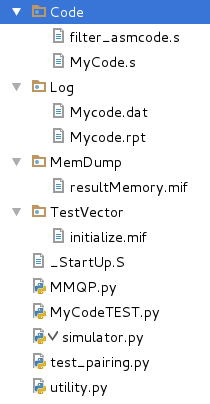
使用python语言实现一个能够测试汇编代码的仿真器

工程文件的说明如下：



过滤后的汇编代码

编译器生成的汇编代码

包含每条指令的日志文件

统计指令出现的频率

存储memory结果

Memory初始化文件

初始化的汇编代码

实现MM算法

汇编代码文件

定义仿真器文件

测试结果文件

定义用到的几个相关函数

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。