决定程序流程的程序计数器

多个内存地址中存储着许多个不同的数据和机器语言指令

例如：
内存地址1000-1005存储程序的指令（机器语言）
内存地址1006-1010存储程序执行时所需要的数据（机器语言）

- 内存地址本身是第2种类型数据
- 内存地址存储的数据是第1种类型数据
- （此处的第1种第2种是作者的叫法，具体可参考02-2种数据....xmind。现实中应该并没有这种叫法，叫法不重要，关键是要理解）

流程控制方式1：顺序执行（累加）

当程序执行时，程序计数器会将其内置数值设定为与起始内存地址相同的数字

例如：
（程序起始）内存地址：1000
内存中的内容：'一段机器语言'表达的指令

程序计数器开始计数，初始值为1000
每执行一个指令 程序计数器+1

流程控制方式2：条件分支（跳转）

在顺序执行的基础上，如果某个内存地址中存储的指令是以下形式：'如果达成某一条件，则跳转到内存地址xxxx'
也就是根据运算结果来判断下一步的内存地址
这时，程序计数器会直接跳转到对应内存地址

例如：
内存地址：1000
内存中的内容：当运算结果大于零时，跳转到1004
（在此处，运算则是由累加寄存器完成的）

而在上例中由累加寄存器计算出的运算结果（此处为比较运算）会被记录在标志寄存器中
程序计数器在跳转内存地址前会根据标志寄存器某位的值来判断是否跳转

例如：
标志寄存器的前3位分别代表'正，零，负'
当其中的某1位为1时，即可判断运算结果的正负
（标志寄存器记录对象：比较运算结果，溢出结果，奇偶校验结果）

小贴士：
程序中的比较指令就是在CPU内部做减法运算

流程控制方式3：循环（反复跳转，直至打破条件）

循环只是1种特殊的条件分支，略

流程的特殊控制方式：函数调用（通过程序计数器的值设定成函数的储存地址来调用函数）

单纯的跳转不能实现调用函数
调用函数后，处理流程应返回'函数调用点'

（请注意！'函数调用点'特指函数存储地址的下一个地址，而不是函数的入口地址）

小贴士：C语言编译成机器语言时，往往一行代码会变成多行机器语言，从而导致这些机器语言离散分布在内存地址中

call指令
1.将'函数调用点'存储在栈中
2.调用并进入函数控制流程
（也就是将函数的入口地址设定给程序计数器）

在将函数的入口地址设定到程序计数器之前（也就是调用函数之前）
call指令已经先把调用后要执行的指令地址（函数存储地址的下一个地址）存储在名为'栈'的主内存里了

return指令
1.将之前保存在栈中的'函数调用点'设定给程序计数器
2.跳转到函数存储地址的下一个地址
（也就是'函数调用点'）

而当函数调用完毕后
函数的出口执行return指令
return指令的功能即把之前call指令保存在栈中的地址设定到程序计数器中

小贴士：高级语言被编译成机器语言后，调用函数的处理会被转换成call指令，而函数结束的处理转换成return指令

浅谈其他寄存器

划分主内存上特定的内存区域（实现'数组'数据构造）

内存的实际地址

基址寄存器 '数组名' 固定值 内存的起始地址 例如：10000000

变址寄存器 '索引' 可变值 内存的相对地址 例如：00000000~0000FFFF

CPU使用两个寄存器时，连续查看内存地址会比用一个寄存器更方便~

小贴士：如果用8位16进制数划分内存，则该范围的数可以通过1个32位的2进制寄存器完成对全部数值的查看

（为了便于理解，虽然左边强调了2进制，但其实寄存器只能存储以2进制形式表示的数据哦。所以无论是多少位的寄存器，都是存储2进制形式，也就是每一位只能是0或者1的数据）

小贴士：关于进制的转换
4位2进制数（0000~1111）
即为1位16进制数(0~F)
3位2进制数（000~111）
则是1位8进制数（0~7）

因为寄存器是2进制
所以要会2进制~

因为内存地址通常用16进制划分
所以要会16进制~

小贴士：什么是数组？
同样长度（其实就是位数）的数据在内存中连续排列的数据构造
数组的单位是'元素'，其构成为'数组名'+'索引'
其中，'数组名'表示全体数据
'索引'用来区分数组中的各个数据

例如：一个10个元素的数组a，'数组a'代表全体数据；'元素a[0]~a[9]'表示其中的各个数据；[]内的数字'0~9'是索引