**PartC**

该部分需要结合前两部分所学，在可以自己添加新指令的基础上尽可能地优化ncopy.ys中的代码，减少其CPE（每元素周期数），是表示指令执行时间随数据量变化快慢的数值。简单来说，就是在实现ncopy函数的前提下，尽可能地减少指令。本次实验做到的优化点包括如下：

1. 添加iaddq指令

因为在代码中有很多常数和寄存器的运算操作，需要irmovq + addq两步实现，使用iaddq一步即可实现，因此类似PartB中所做的，在pipe-full.hcl中添加iaddq指令，并将ncopy.cy中相关的代码修改即可。

1. 减少andq指令

在程序中经常需要修改rdx(len)的值，之后再判断的情况，原始代码实现方式举例如下：

**iaddq $-1, %rdx**

**andq %rdx, %rdx**

**jle Done**

其中andq指令是多余的，因为%rdx刚刚参与过运算，因此可以删除这类andq指令。

1. 循环展开

循环展开在书中有提到过，实验中采用了6×1展开(写更多的展开应该还可以小幅度地提升大规模数据的执行效率)，即每次循环完成六个值的复制。剩下的值最多为5个，按序复制直到len值为0(即没有还需要复制的值)。

1. 避免加载/使用冒险

在6×1的展开中，会有多组如下代码：

**…………**

**mrmovq x(%rdi), %r10**

**rmmovq %r10, x(%rsi)**

**…………**

为了避免加载/使用冒险，rm指令需要等到mr指令完成回写才能执行，即这两条指令之间的流水会留出一定的空隙，因此可以在中间插入一条后面需要执行的命令来利用这一段空隙，利用方式如下：

**…………**

**mrmovq x(%rdi), %r10**

**mrmovq x+8(%rdi), %r11**

**rmmovq %r10, x(%rsi)**

**…………**

**rmmovq %r11, x+8(%rdi)**

**…………**

如代码所示，每两组mrmovq+rmmovq指令结合在一起，将后一组的mrmovq指令移到前一组的两个指令之间，用后面需要的计算来填补前面产生的空隙，从而节省时间。

根据上述四种优化，最终的平均CPE为8.26，成绩为44.9/60，以后如果有时间再考虑如何进一步优化。