# 混淆指令序列分析

分析以下混淆汇编指令片段所实现的功能。请分以下两种情况解答：

(1)若 addr\_branch\_default 为 ret 下一条指令的地址, 以下混淆结果等价于什么指令？

(2)若 addr\_branch\_default 为任一其他指令地址, 以下混淆结果等价于什么指令序列？

push addr\_branch\_default

push ebx

push edx

mov ebx, [esp+8]

mov edx, addr\_branch\_jmp

cmovz ebx, edx ;条件 mov, 若 ZF=1 则 mov, 否则相当于 NOP

mov [esp+8], ebx

pop edx

pop ebx

ret

用文字描述每条指令执行后的栈状态和关键寄存器状态。对以上每一问，从 cmovz 开始进一步分情况讨论，最后归纳讨论结果，得出每一问的结论指令或指令序列。

**原有指令执行后的栈状态和关键寄存器状态**：

**push addr\_branch\_default:**

将地址 addr\_branch\_default 压入栈顶。

将地址值推送到栈上，通常用于将地址或常量值推送到栈上，以备后续使用。

**push ebx:**

将寄存器 ebx 的值压入栈顶。

将 ebx 寄存器的值保存到栈上，通常用于保存寄存器状态。

**push edx:**

将寄存器 edx 的值压入栈顶。

将 edx 寄存器的值同样保存到栈上。

**mov ebx, [esp+8]:**

将栈顶偏移 8 字节处的值加载到 ebx 寄存器中。

从栈上弹出一个值，然后将其存储到 ebx 寄存器中。

**mov edx, addr\_branch\_jmp:**

将地址 addr\_branch\_jmp 的值加载到 edx 寄存器中。

将一个地址值加载到 edx 寄存器中，通常用于设置寄存器的值以备后续使用。

**cmovz ebx, edx:**

执行条件移动，如果 "零标志位" (ZF) =1，则将 edx 寄存器的值复制到 ebx 寄存器中。

根据 ZF 标志的状态来确定是否执行移动操作。若 ZF=1 则 mov, 否则相当于 NOP

**mov [esp+8], ebx:**

将 ebx 寄存器的值存储到栈顶偏移 8 字节处。

将 ebx 寄存器的值写回到栈上的一个特定位置。

**pop edx:**

将栈顶的值弹出并存储到 edx 寄存器中。

从栈中弹出一个值并将其加载到 edx 寄存器中。

**pop ebx:**

将栈顶的值弹出并存储到 ebx 寄存器中。

还原先前保存的寄存器状态。

**分情况讨论指令序列**：

**情况 1**: addr\_branch\_default 为 ret 指令地址

1. cmovz ebx, edx：

如果 ZF =1，即相等条件成立，ebx 寄存器将被设置为 edx 的值。

1. mov [esp+8], ebx：

将 ebx 寄存器的值写回到栈上偏移 8 字节处。

1. pop edx 和 pop ebx：

将栈顶的值分别弹出并存储到 edx 和 ebx 寄存器中，即恢复之前保存的寄存器状态。

1. ret：

从当前函数返回，跳转到栈顶的地址（ret 指令的地址），完成函数调用。

**情况 2**: addr\_branch\_default 为任意其他指令地址

1. cmovz ebx, edx：

如果 ZF =1，即相等条件成立，ebx 寄存器将被设置为 edx 的值。

1. mov [esp+8], ebx：

将 ebx 寄存器的值写回到栈上偏移 8 字节处。

1. pop edx 和 pop ebx：

将栈顶的值分别弹出并存储到 edx 和 ebx 寄存器中，即恢复之前保存的寄存器状态。

1. jmp addr\_branch\_default：

跳转到 addr\_branch\_default 地址处执行相应的指令。

当 addr\_branch\_default 为 ret 指令地址时，混淆的结果等价于将 cmovz 指令以及之后的一系列指令转化为

pop ebx

pop edx

ret

当 addr\_branch\_default 为任意其他指令地址时，混淆的结果等价于将 cmovz 指令以及之后的一系列指令转化为

pop ebx

pop edx

jmp addr\_branch\_default