1、
解析：
F:取指阶段由图可知需要再取出 1 个 byte 表示两个寄存器，然后计算 valP
D:因为第一个寄存器不存在，所以只需要取出第二个寄存器的值
E:执行阶段实际上不需要进行操作，但是为了满足 ALU，需写成 valE <- valB +0，同时由 PC 阶段知道此时还要设置条件码
M:None
W:None

2、 不需要。 解析：增加转发路径后，间接跳转在 decode 阶段时 SelectPC 正好利用最新转发的 valB 的值作为预测地址访问指令内存。

D_icode == IJREGXX : d_valB， ①②③。解析：1:F_predPC 的判定条件为 1，即只要运行到此处就会选择该分支，故④是无效的插入位置。注意，原来的 M_XXX 条件和 W_XXX 不会同时发生（当 ret 在 W 阶段时，D/E/M 阶段为插入的 bubble），所以其顺序任意，并且它们都不会和新加入的条件冲突。ret 的情况同上；预测错误发生时，Decode 阶段的指令已经被清空，不会出现 D_icode==IJREGXX。所以最终不会导致多个条件成立，即加入的触发条件只要在 1:F_predPC 前就可以。
word f_pc = [
①// D_icode == IJREGXX: d_valB;
(M_icode == IJXX || M_icode == IJREGXX) && !M_cnd : M_valA;
②// D_icode == IJREGXX: d_valB;
W_icode == IRET : W_valM;
③//D_icode == IJREGXX: d_valB;
1 : F_predPC;
④
]

3、 E_icode == IJREGXX && !e_Cnd

F	D	E	M	W
normal/stall	bubble	bubble	normal	normal

解析：分析方法同 IJXX。注意，触发条件是在执行阶段，因为执行阶段会通过 ALU 计算出 e_cnd 得出是否需要跳转。此时 IRJREGXX 在 E 阶段，故条件为 E_icode == IJREGXX && !e_Cnd。同时处理方式同 IJXX 类似，都需要添加气泡取消错误执行的两条指令。

4、 15 13 20
解析：
注意，计算直到返回指令 ret 完全通过流水线为止，故 ret 所需周期数为（1+4）
下面分析 n 取不同值时的执行情况：
n == -1: line7 时 rdi 为-1，与操作之后设置条件码 SF（最近的操作结果为负数），line 8 分支预测错误惩罚 2 个周期。故需 9 + 2 + 4 = 15。
n == 0: line7 时 rdi 为 0，line8 可以正常跳转，故需 9 + 4 = 13。

n == 1:　line7 时 rdi=1，line8 可以正常跳转。line 12 和 13 发生加载/使用冒险，惩罚 1 个周期。故需 12 + 1 + 4 = 17。

附各冒险判断条件：

条件	触发条件
处理 ret	IRET∈{D_icode,E_icode,M_icode}
加载/使用冒险	E_icode∈{IMRMOVL,IPOPL} & & E_dstM∈{d_srcA,d_srcB}
预测错误的分支	E_icode=IJXX & & ! e_Cnd
异常	m_stat∈{SADR,SINS,SHLT}||W_stat∈{SADR,SINS,SHLT}

n == 2:　line7 时 rdi=2，line8 可以正常跳转。line 12 和 line 13，以及 line 10 和 line 11 各发生一次加载数据冒险，共惩罚 2 个周期。一共 14 + 2 + 4 = 20。