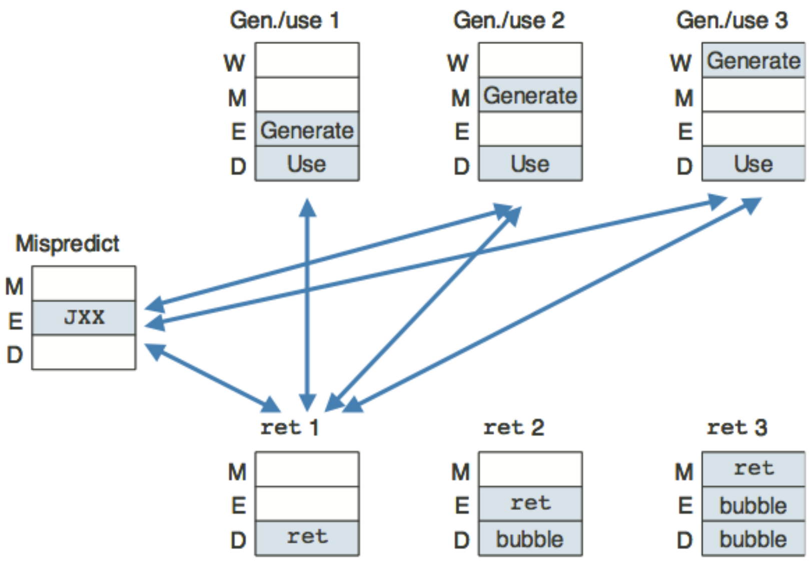
4.51



This is a hard problem, because there are many possible combinations of special cases that can occur simultaneously. The figure above illustrates this problem. We can see that there are now three variants of generate/use cases, where the instruction in the execute, memory, or write-back stage is generating a value to be used by the instruction in the decode stage. The second and third generate/use cases can occur in combination with a mispredicted branch. In this case, we want to handle the misprediction, injecting bubbles into the decode and execute stages.

For cases where a misprediction does not occur, each of the generate/use conditions can occur in combination with the first ret pattern (where ret uses the value of %esp). In this case, we want to handle the data hazard by stalling the fetch and decode stages and injecting a bubble into the execute stage.

```
# Should I stall or inject a bubble into Pipeline Register F?
# At most one of these can be true.
bool F_bubble = 0;
bool F_stall =
        # Stall if either operand source is destination of
        # instruction in execute, memory, or write-back stages
        d_srcA != RNONE && d_srcA in
          { E_dstM, e_dstE, M_dstM, M_dstE, W_dstM, W_dstE } ||
        d_srcB != RNONE && d_srcB in
          { E_dstM, e_dstE, M_dstM, M_dstE, W_dstM, W_dstE } ||
        # Stalling at fetch while ret passes through pipeline
        IRET in { D_icode, E_icode, M_icode };
```

```
# Should I stall or inject a bubble into Pipeline Register D?
# At most one of these can be true.
bool D_stall =
        # Stall if either operand source is destination of
        # instruction in execute, memory, or write-back stages
        # but not part of mispredicted branch
        !(E_icode == IJXX && !e_Cnd) &&
         (d_srcA != RNONE && d_srcA in
            { E_dstM, e_dstE, M_dstM, M_dstE, W_dstM, W_dstE } ||
          d_srcB != RNONE && d_srcB in
            { E_dstM, e_dstE, M_dstM, M_dstE, W_dstM, W_dstE });

bool D_bubble =
        # Mispredicted branch
        (E_icode == IJXX && !e_Cnd) ||
        # Stalling at fetch while ret passes through pipeline

        # but not condition for a generate/use hazard
        !(d_srcA != RNONE && d_srcA in
           { E_dstM, e_dstE, M_dstM, M_dstE, W_dstM, W_dstE } ||
          d_srcB != RNONE && d_srcB in
           { E_dstM, e_dstE, M_dstM, M_dstE, W_dstM, W_dstE }) &&
          IRET in { D_icode, E_icode, M_icode };

# Should I stall or inject a bubble into Pipeline Register E?
# At most one of these can be true.
bool E_stall = 0;
bool E_bubble =
        # Mispredicted branch

        (E_icode == IJXX && !e_Cnd) ||
          # Inject bubble if either operand source is destination of
          # instruction in execute, memory, or write back stages
          d_srcA != RNONE &&
            d_srcA in { E_dstM, e_dstE, M_dstM, M_dstE, W_dstM, W_dstE } ||
          d_srcB != RNONE &&
            d_srcB in { E_dstM, e_dstE, M_dstM, M_dstE, W_dstM, W_dstE };
```

4.54

This problem requires changing the logic for predicting the PC value and the misprediction condition. It requires distinguishing between conditional and uncondiational branches.

```
## What address should instruction be fetched at
int f_pc = [
        # Mispredicted branch.  Fetch at incremented PC
        # BNT: Changed misprediction condition
        M_icode == IJXX && M_ifun != UNCOND && M_Cnd : M_valE;
        # Completion of RET instruction.
        W_icode == IRET : W_valM;
        # Default: Use predicted value of PC
        1 : F_predPC;
```

```
# Predict next value of PC
int f_predPC = [
        # BNT: Revised branch prediction rule:
        #   Unconditional branch is taken, others not taken
        f_icode == IJXX && f_ifun == UNCOND : f_valC;
        f_icode in { ICALL } : f_valC;
        1 : f_valP;
];
```

```
## Select input A to ALU
int aluA = [
        E_icode in { IRRMOVL, IOPL } : E_valA;
        # BNT: Use ALU to pass E_valC to M_valE
        E_icode in { IIRMOVL, IRMMOVL, IMRMOVL, IJXX } : E_valC;
        E_icode in { ICALL, IPUSHL } : -4;
        E_icode in { IRET, IPOPL } : 4;
        # Other instructions don't need ALU
];

## Select input B to ALU
int aluB = [
        E_icode in { IRMMOVL, IMRMOVL, IOPL, ICALL,
                     IPUSHL, IRET, IPOPL } : E_valB;
        # BNT: Add 0 to valC
        E_icode in { IRRMOVL, IIRMOVL, IJXX } : 0;
        # Other instructions don't need ALU
];
```

```
bool D_bubble =
        # Mispredicted branch
        # BNT: Changed misprediction condition
        (E_icode == IJXX && E_ifun != UNCOND && e_Cnd) ||
        # Stalling at fetch while ret passes through pipeline
        # but not condition for a load/use hazard
        !(E_icode in { IMRMOVL, IPOPL } && E_dstM in { d_srcA, d_srcB }) &&
          IRET in { D_icode, E_icode, M_icode };
```

```
bool E_bubble =
        # Mispredicted branch
        # BNT: Changed misprediction condition
        (E_icode == IJXX && E_ifun != UNCOND && e_Cnd) ||
        # Conditions for a load/use hazard
        E_icode in { IMRMOVL, IPOPL } &&
         E_dstM in { d_srcA, d_srcB};
```

4.57

A. Here's the formula for a load/use hazard:

$$\text{E\_icode} \in \{\text{IMRMOVL}, \text{IPOPL}\} \ \&\& \ (\text{E\_dstM} = \text{d\_srcB} \ || \ \text{E\_dstM} = \text{d\_srcA} \ \&\& \ !\text{D\_icode} \in \{\text{IRMMOVL}, \text{IPUSHL}\})$$

B. The relative HCL code for the control logic is shown below:

```
int e_valA = [
        # Forwarding Condition
        M_dstM == E_srcA && E_icode in { IPUSHL, IRMMOVL } : m_valM;
        1 : E_valA;  # Use valA from stage pipe register
];

bool F_stall =
        # Conditions for a load/use hazard
        E_icode in { IMRMOVL, IPOPL } &&
         (E_dstM == d_srcB ||
          (E_dstM == d_srcA && !D_icode in { IRMMOVL, IPUSHL })) ||
        # Stalling at fetch while ret passes through pipeline
        IRET in { D_icode, E_icode, M_icode };

# Should I stall or inject a bubble into Pipeline Register D?
# At most one of these can be true.
bool D_stall =
        # Conditions for a load/use hazard
        E_icode in { IMRMOVL, IPOPL } &&
        E_icode in { IMRMOVL, IPOPL } &&
         (E_dstM == d_srcB ||
          (E_dstM == d_srcA && !D_icode in { IRMMOVL, IPUSHL }));

bool E_bubble =
        # Mispredicted branch
        (E_icode == IJXX && !e_Cnd) ||
        # Conditions for a load/use hazard
        E_icode in { IMRMOVL, IPOPL } &&
         (E_dstM == d_srcB ||
          (E_dstM == d_srcA && !D_icode in { IRMMOVL, IPUSHL }));
```