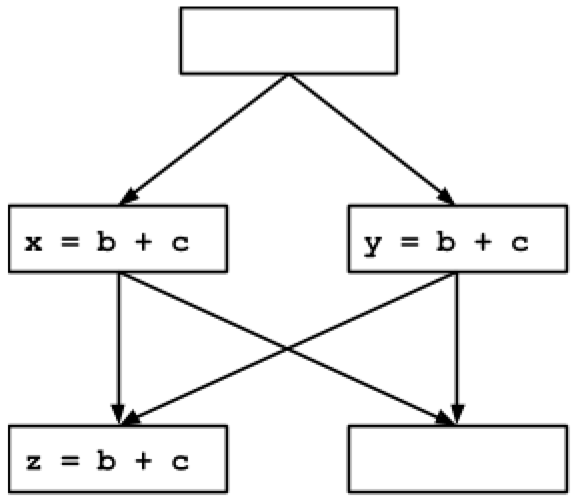
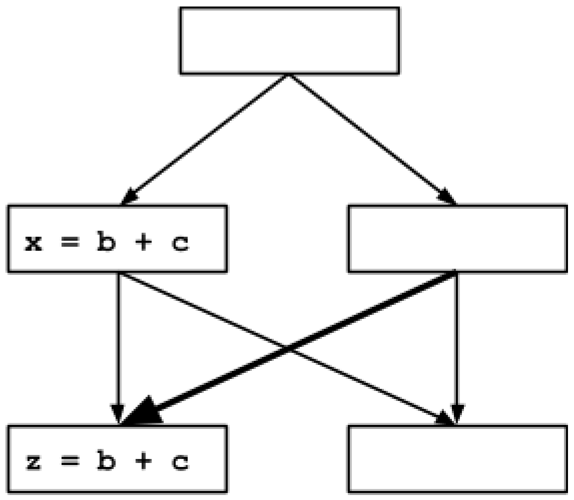
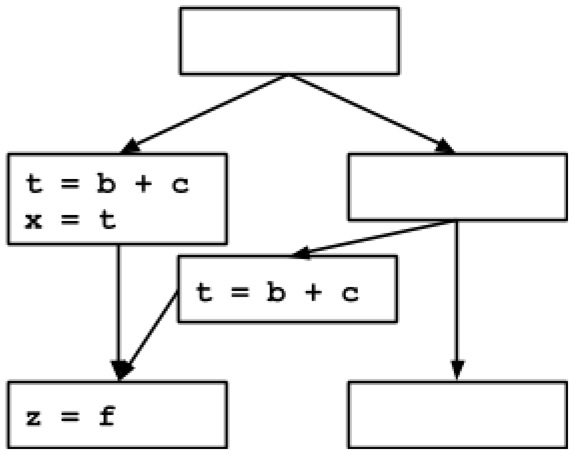
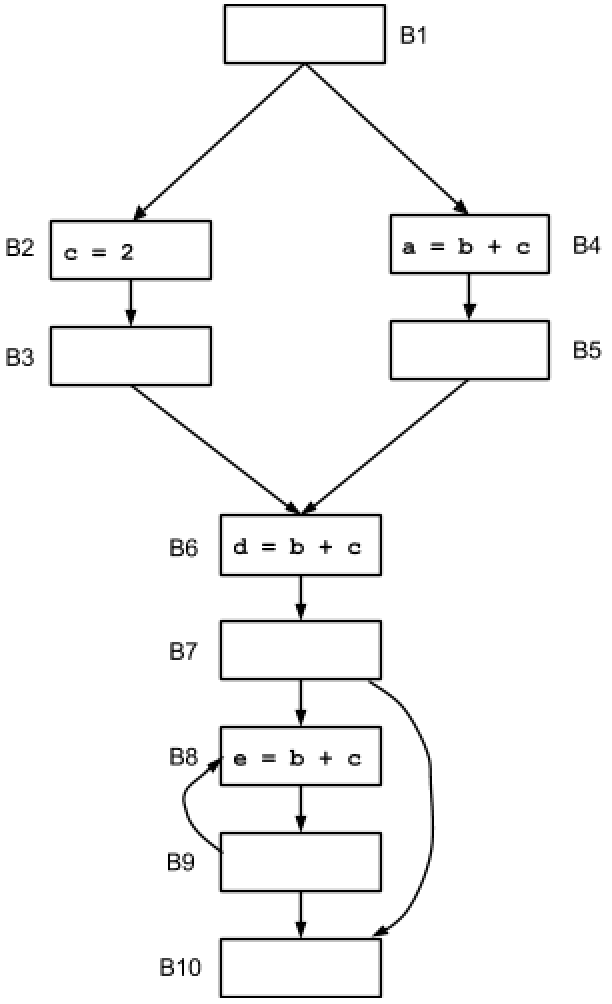
Full Redundancy vs Partial Redundancy

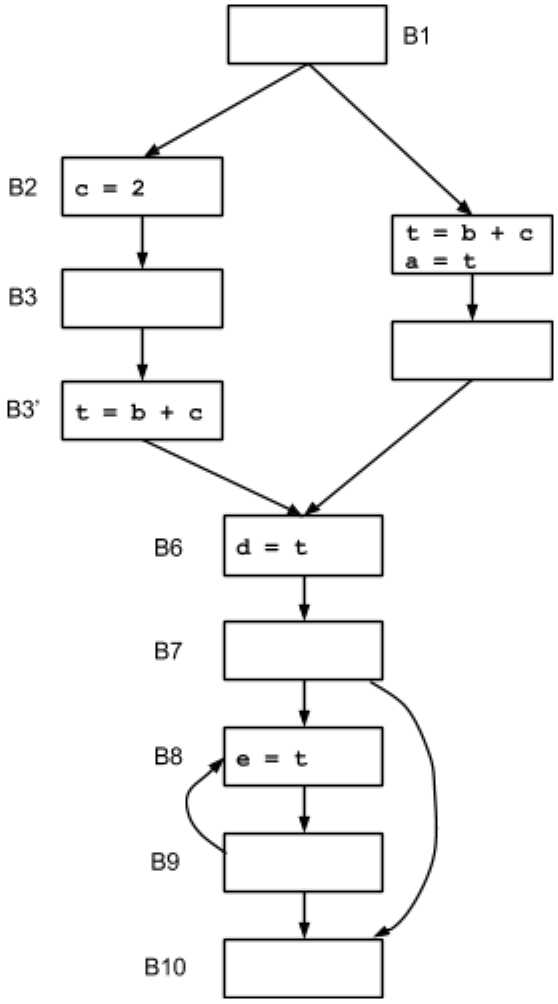






Lazy code motion example 1

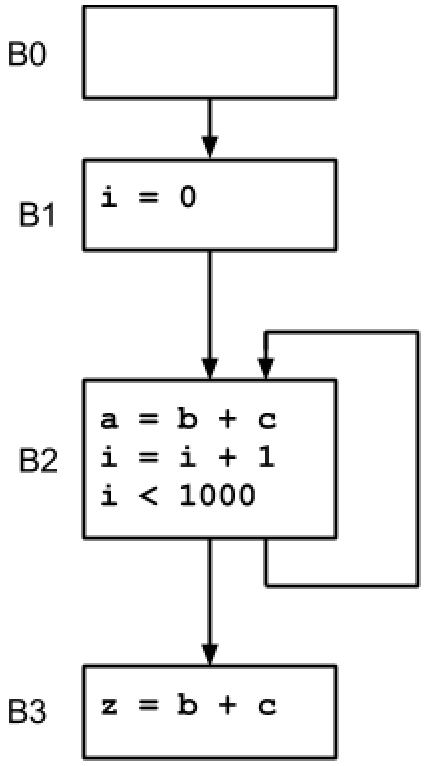


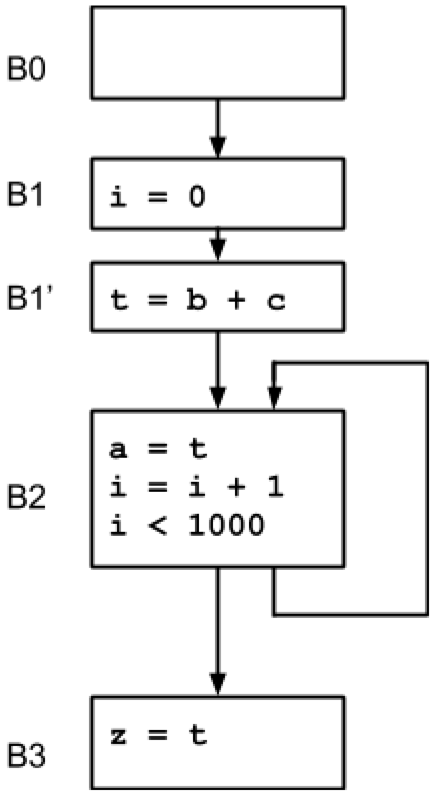


Expression `b + c` is :

- Anticipated in blocks B3, B3' (new block after B3), B4, B5, B6, B8. It is not anticipated on entry to B2 since value of c is recomputed. It is not anticipated at B1 because it is not required on branch B1 to B2. It is not anticipated at the beginning of B7 because of branch B7 to B10 where it is not required. Anticipation of an expression can oscillate along a path. For instance, b+c is anticipated at B6, not anticipated at B7 and anticipated at B8.
- Available in all points except B1, beginning of B2, beginning of B3, and beginning of B4. We can make the expression available earliest in B3 and B4.
- b + c can be postponed at B3 and at beginning of B3' to B3', and at beginning of B4 to B4. We cannot postpone b+c beyond B4 since it is used in B4. We cannot postpone it beyond B3' on the left branch as postponing it to B6 will lead to redundancy -- b + c is not postponable from one of the predecessors of B6 to B6.

Lazy code motion example 2

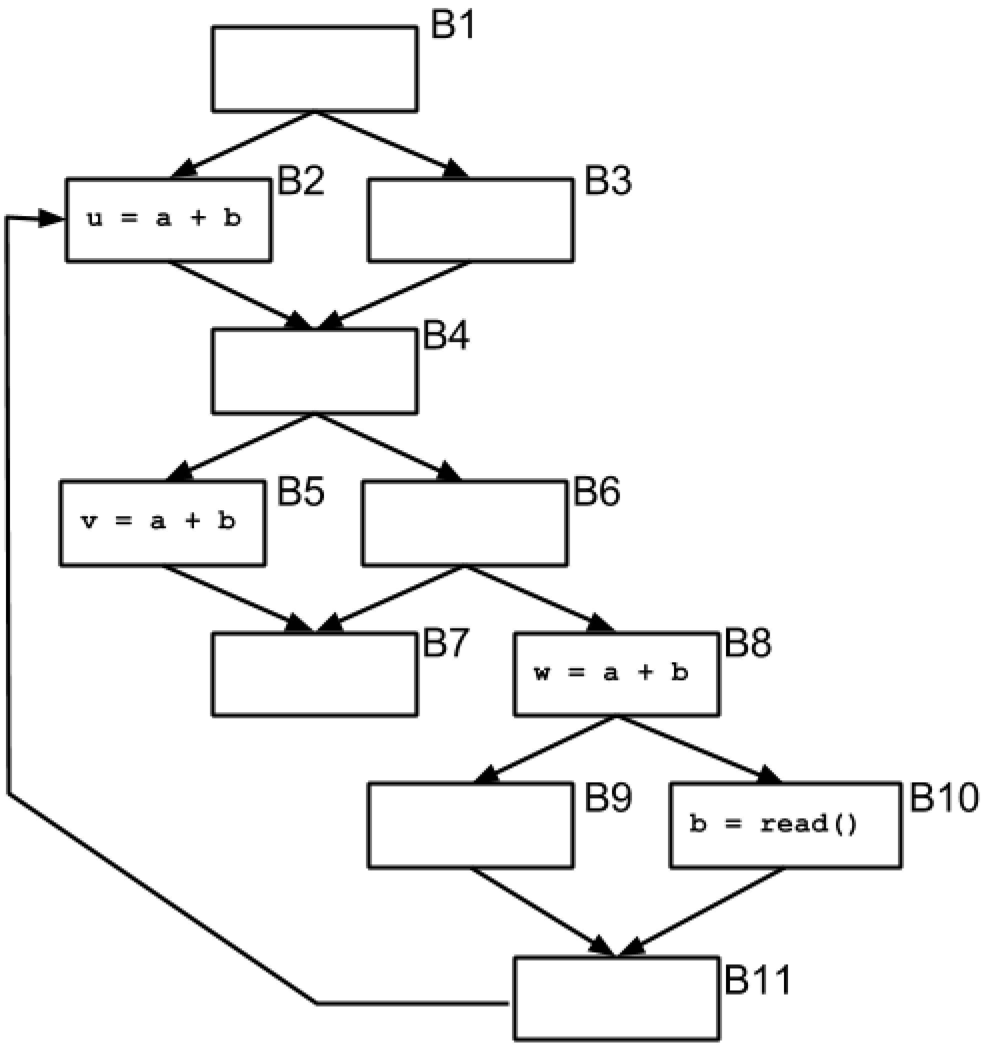


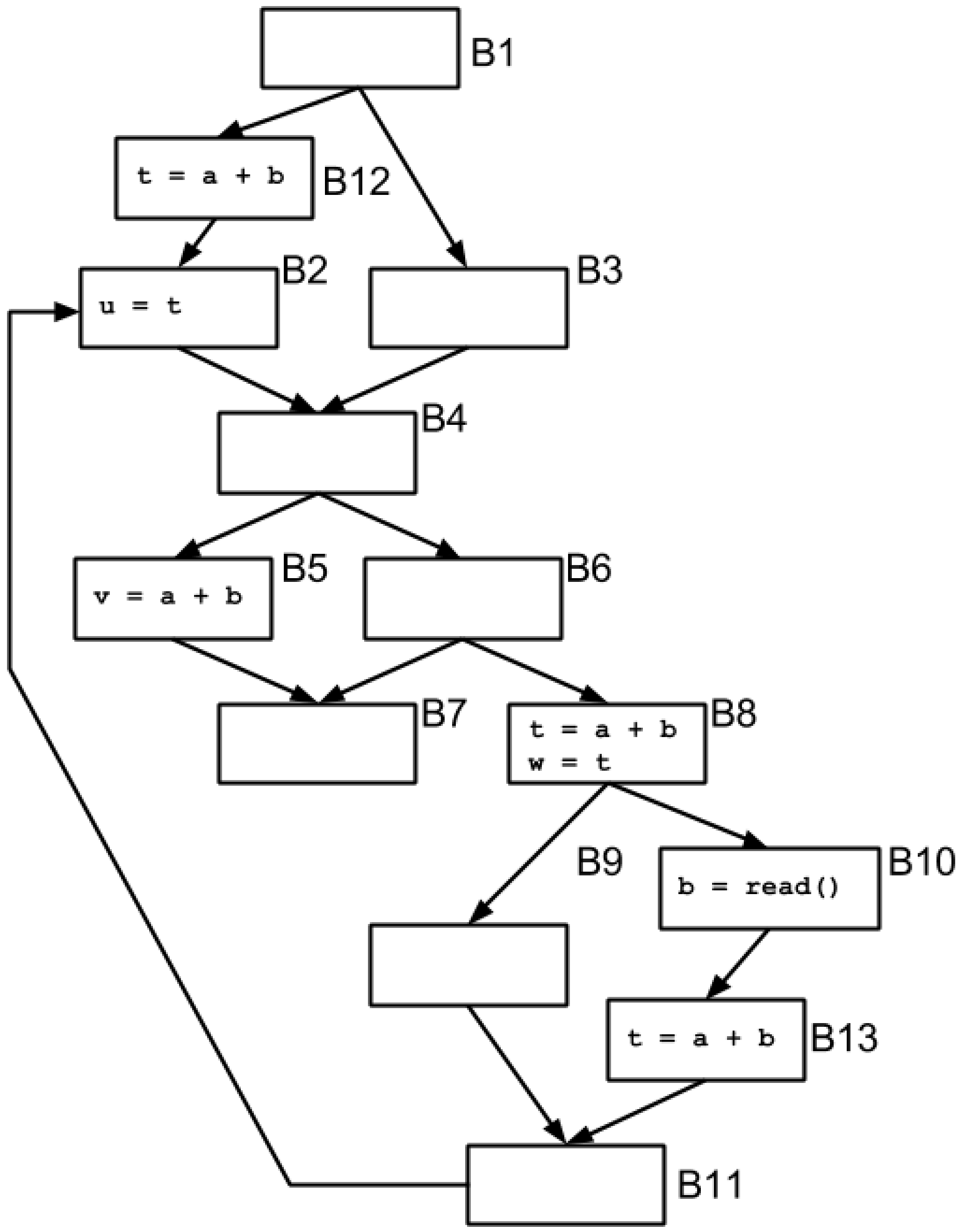


Expression b + c is:

- Anticipated at B0, B1, B1’ (block after B1), B2, B3
- Available (available-able) at after-B0, B1, B2, B3
- Can be earliest computed at B0
- Is postponable to B0, B1, and B1’, but not beyond (since B2 is B2’s predecessor, and B2 uses b + c)
- Needs to be computed latest at B1’
- Is used beyond B1’

Lazy code motion example 3



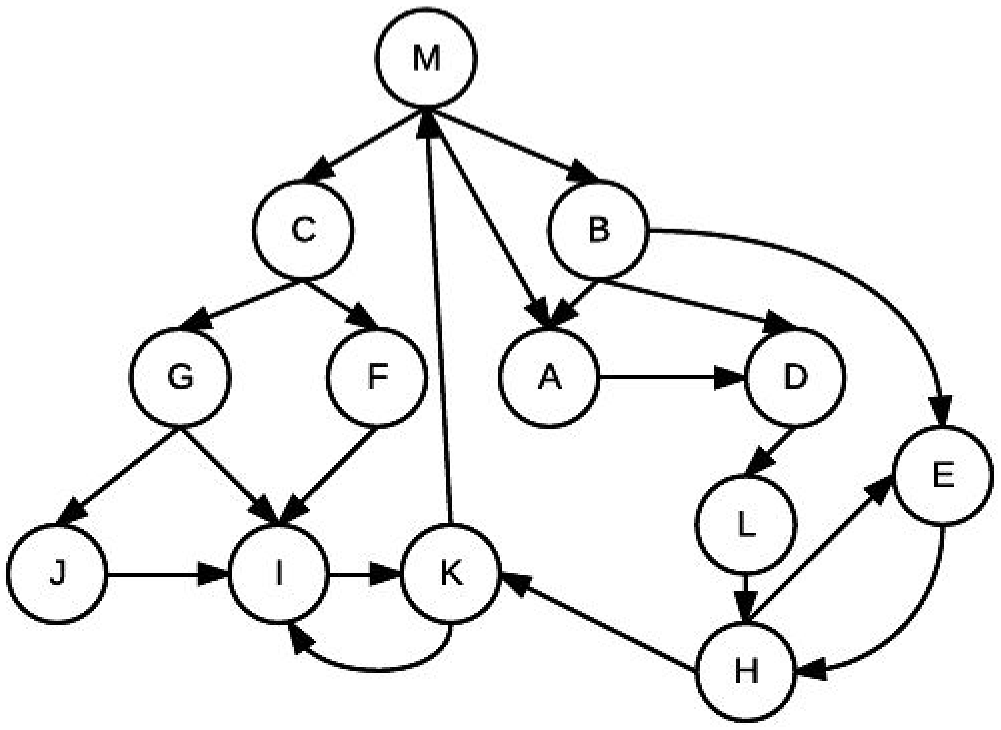


- Add B12 between B1 and B2, and add B13 between B10 and B13
- Cannot move a+b from B5 up. Why? Because B1 → B3 → B4 → B6 → B7 is a valid path to exit. Moving a+b up from B5 will introduce a computation along a path where it would have not existed in the original program. The expression a + b is not anticipated along B1, B3, B4, B6, B7, which means we will definitely not put a + b there.
- v = a + b is not used beyond B5, so temporary variable is removed.
- Similarly, a + b in B8 cannot be lifted any further, as it is not anticipated at B6 or B1.

- The transformed code has more a + b expressions than the original code. But if you follow a path, you actually compute a + b the same number of times or less. Example of less: B8 → B9 → B11 → B2. Example of equal: B10 → B11 → B2 (or B10 → B13 → B11 → B2 in transformed code).
- Importantly, the code does not compute b + c more number of times when it is run.

Dominators

Draw the dominator tree for the following control flow graph. What are the natural loops in this flow graph?


M
C
B
G
F
A
D
E
L
J
I
K
H