Exercises on Analysis and Optimisation

With solutions

1. Consider the following program.

```
int foo(int e) {
  int a, b, c, d;
  a = 1;
  b = 2;
  do {
    c = a + b;
    d = c - a;
    while (e != 0) {
      d = b & d;
      if (d == 0) break;
      d = a + b;
      e = e + 1;
    }
    b = a + b;
    e = c - a;
  } until (b != 0);
  a = b & d;
  b = a - d;
  return b;
}
```

[0] I'd like to thank Jacky Jiang for finding and fixing some errors in an earlier version of this document.

(a) Convert the program into 3-address code. You may use any reasonable 3-address instructions.

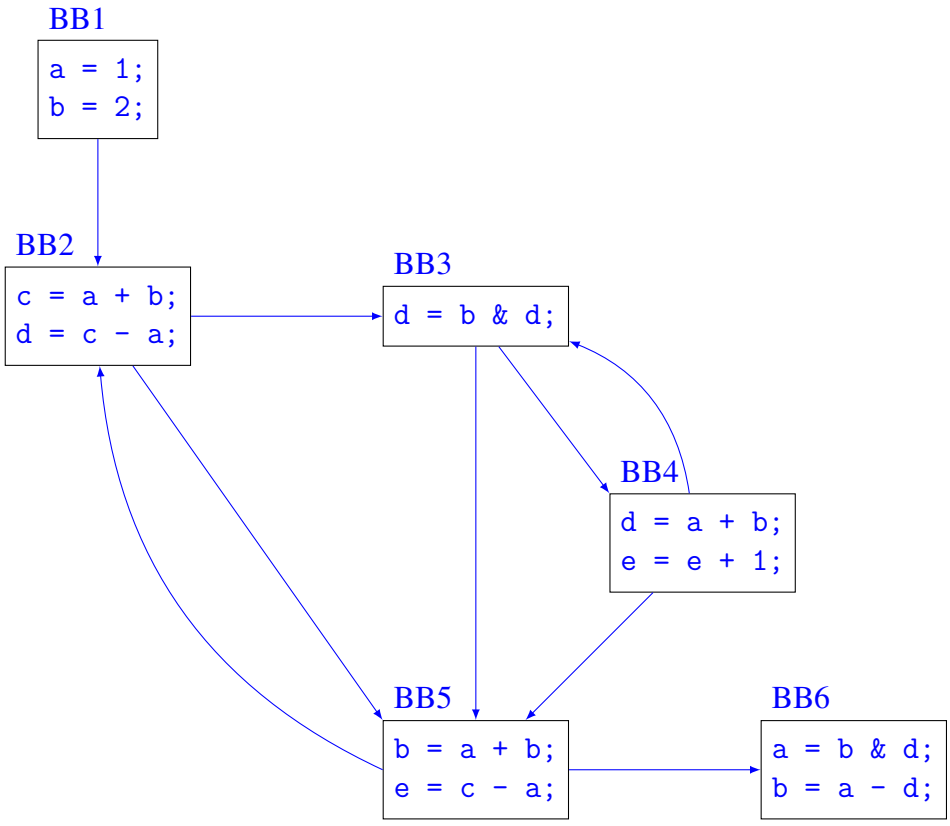
Solution:

```
foo:
L1: a = 1;
    b = 2;
L2: c = a + b;
    d = c - a;
    ifzero e goto L5;
L3: d = b & d;
    ifzero d goto L5;
L4: d = a + b;
    e = e + 1;
    ifnotzero e goto L3;
L5: b = a + b;
    e = c - a;
    ifzero b goto L2;
L6: a = b & d;
    b = a - d;
```

Note, by the way, that there are two instructions that test whether `e` *is zero. These correspond to the test before the first iteration of the* `while`*-loop and the test after each iteration. Writing my 3-address code like this means that I can avoid one* `jump` *per iteration, so my program's performance may be a little better.*

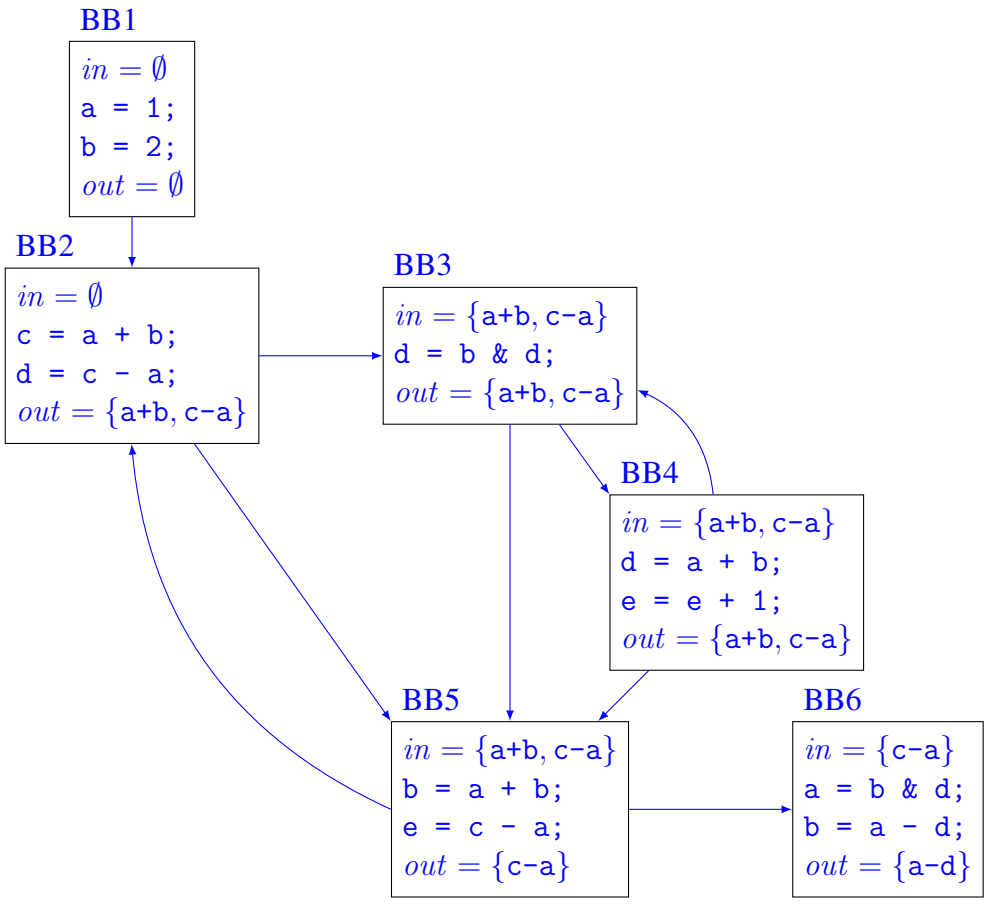
(b) Represent your 3-address program as a flow graph of basic blocks.

Solution: *You might also add explicit 'entry' and 'exit' blocks if you like; I haven't bothered to do so.*

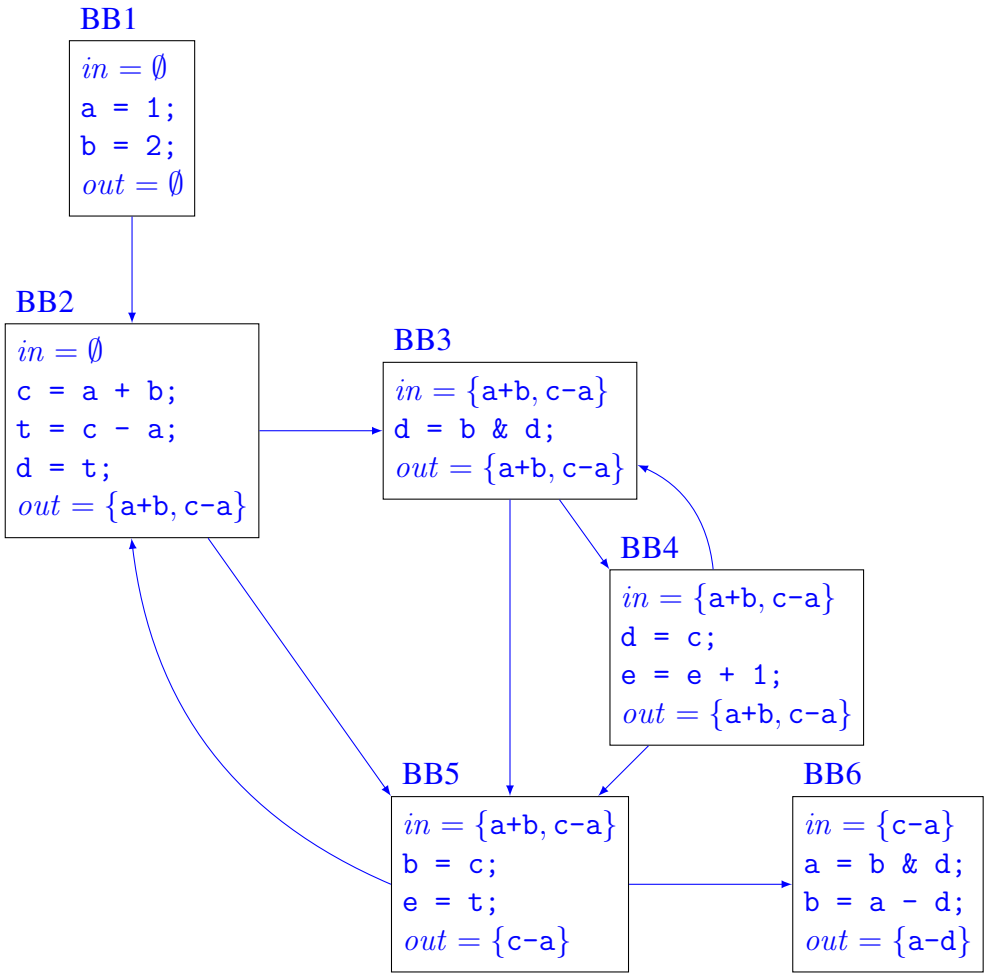


(c) Compute the available expressions at the entry and exit of each basic block. Does your analysis suggest any opportunity for common subexpression elimination? If so, perform this optimisation.

Solution: *The available expressions are as follows.*

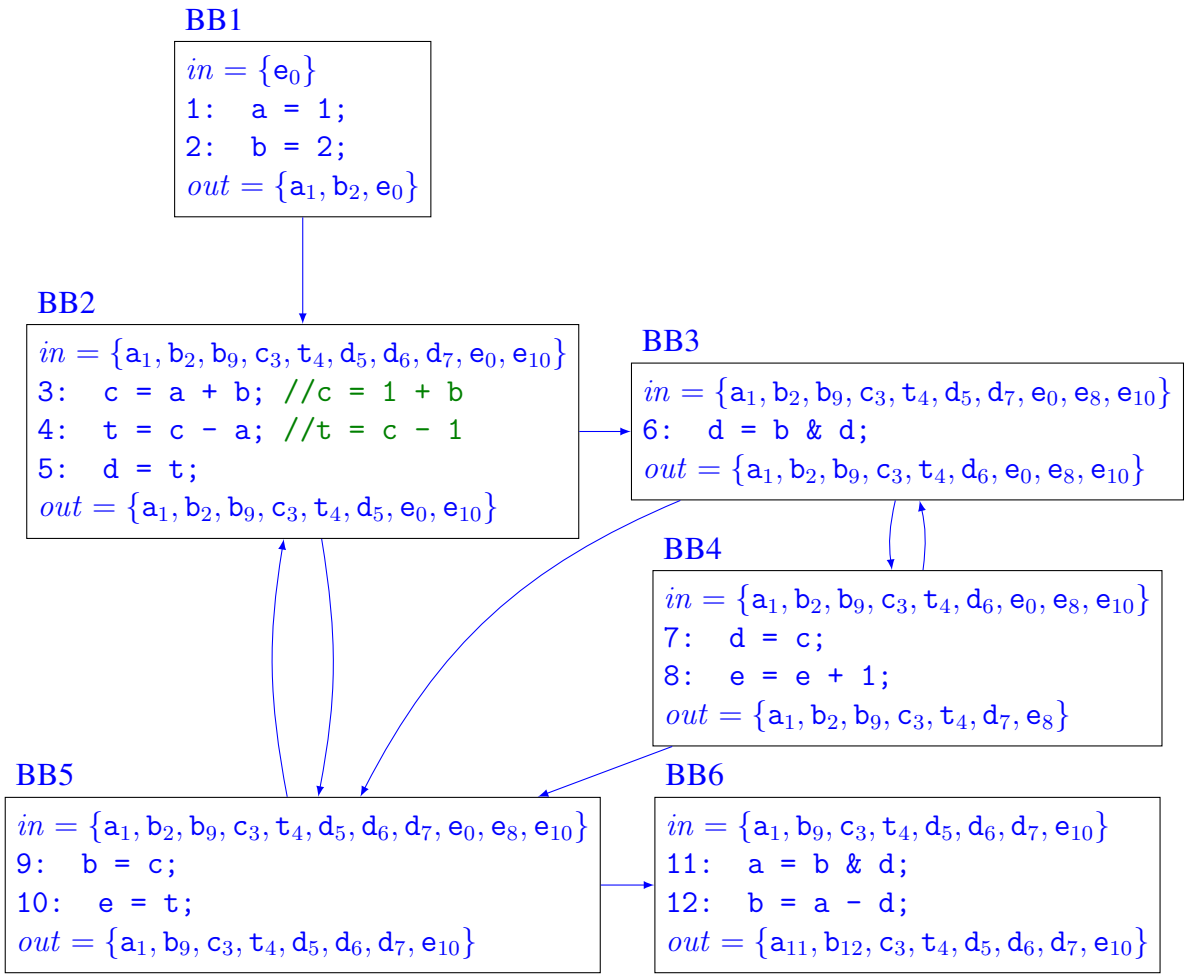


We can then see that `a+b` *is available in BB4 (so* `d = a + b` *can become* `d = c`*) and in BB5 (so* `b = a + b` *can become* `b = c`*). Also,* `c - a` *is available in BB5, but no available variable currently holds it, so we need to create a temporary, say* `t`*. Our flow graph thus becomes:*



(d) Using your optimised flow graph, compute the reaching definitions at the entry and exit of each basic block. Does your analysis suggest any opportunity for constant propagation? If so, perform this optimisation.

Solution: *In the following, I write x_{42} to mean 'the definition of variable x on line 42', and so on.*

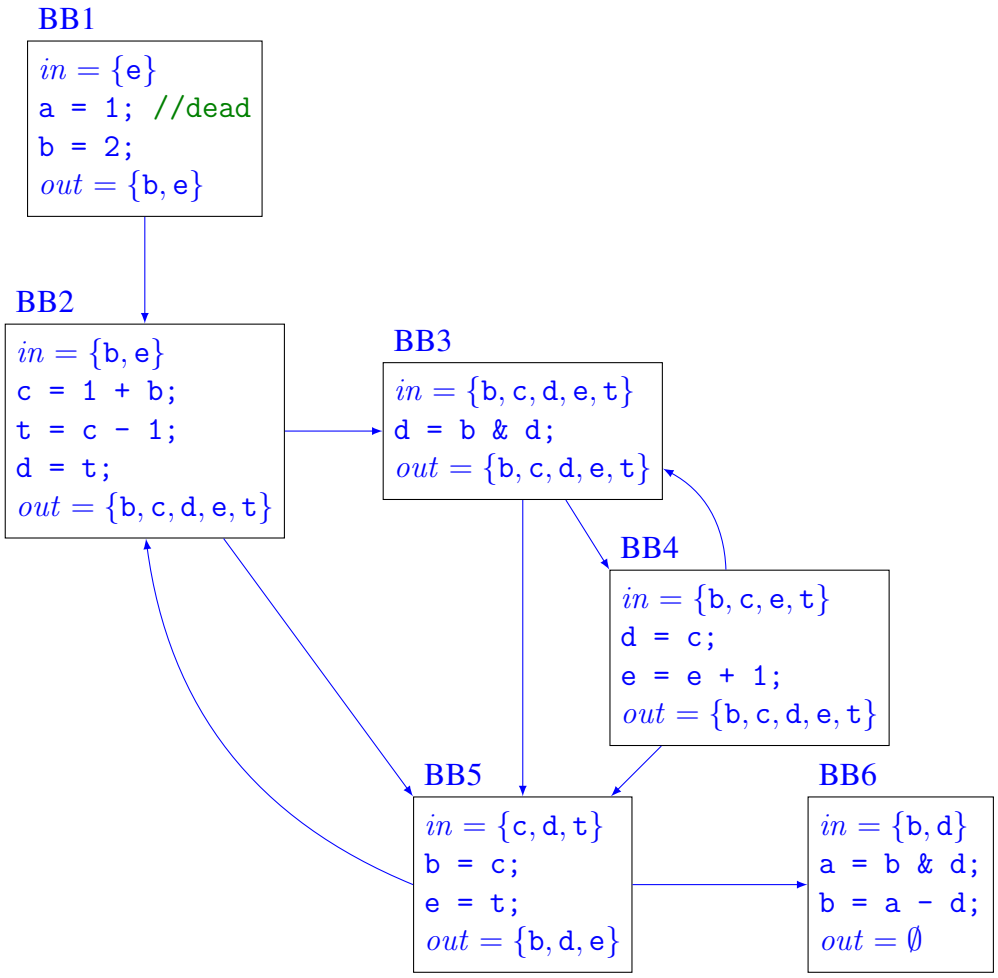


Note, by the way, that the in set for BB1 contains e_0, which stands for the definition of e provided as an argument to the function.

There is an opportunity for constant propagation in BB2: a *can be replaced with 1 on lines 3 and 4, as shown in the green comments.*

(e) Using your optimised flow graph, compute the live variables at the entry and exit of each basic block. Does your analysis suggest any opportunity for dead code elimination? If so, perform this optimisation.

Solution: *The live variables are as follows.*



The assignment marked as 'dead' can be safely eliminated.