9 Optimising Compilers (tmj32)

The following excerpt from a program in C-style code is optimised by a compiler using data-flow analyses and transformations. Assume that variables \( x \), \( y \) and \( z \) have already been defined:

```c
a = x - y
if (a > 3) {
    b = a + z
    c = x - y
} else {
    b = a + z
    a = a * b
}
d = x - y
b = b / d
print(a * b)
```

(a) Using available expression analysis, perform common subexpression elimination on the code showing the results of both the analysis and transformation. [5 marks]

(b) Using very busy expression analysis, perform code hoisting on the code from part (a) showing the results of both the analysis and transformation. [4 marks]

(c) Using reaching definition analysis, perform copy propagation on the code from part (b) showing the results of both the analysis and transformation. [Hint: use the results of the analysis to transform across basic blocks.] [4 marks]

(d) Using live variable analysis, perform dead code elimination on the code from part (c) showing the results of both the analysis and transformation. [4 marks]

(e) Perform if simplification on the code from part (d) showing the result of the transformation. [3 marks]