Optimising Compilers

Summarise the idea of the 3-address instruction and briefly indicate its advantage over stack-oriented instructions as intermediate code. Explain the notion of the flowgraph containing such instructions, giving such a flowgraph for the C function:

```c
int f(int x, int g())
{
    return x==0 ? g(x) : x*f(x-1, g);
}
```

You should explain how arguments and results of functions in your example are communicated; also make clear any difference in the representation of calls to f and g. [8 marks]

Given a flowgraph in which each node contains a single 3-address instruction (not grouped into basic blocks), design dataflow equations (and thence an algorithm) for reaching definitions. [9 marks]

The reaching definitions $RD(n)$ of a node $n$ are the set of nodes $m$ such that $m$ contains a 3-address instruction such as

$$m : x := a + b$$

which writes to ("defines") one of its operands (here $x$), and such that there is a path in the flowgraph from $m$ to $n$ by which the value given to $x$ at $m$ may still be unchanged (by other assignments to $x$) when it reaches $n$. Hence in

1: $y := 1$
2: if $x$<1 goto 5;
3: $y := x*y$
4: $x := x-1$; goto 2;
5: return

we would have $RD(3) = \{1, 3, 4\}$ and $RD(4) = \{3, 4\}$; note that the definition (of $y$) at 3 reaches via the loop back to 3 even though it would not be available because of node 4.

Develop a program in which $m \in RD(n)$ but which no run-time execution can cause the value assigned at $m$ to reach $n$. To what extent can we fix this problem? [3 marks]

[Hint: work by analogy from live variable or available expression analysis; determine the direction of the analysis and suitable gen and kill properties of nodes. You may find it convenient to consider cases like "l contains a statement $x := e$" or "l contains some other statement".]