CS 744 – Advanced Compiler Design – Assignment 3

Part 1 – Partial Redundancy Elimination (20 marks)

In this part of the assignment, you will perform partial redundancy elimination (the variation due to Drechsler and Stadel [1] that was presented in class) on the following program. See the lecture slides from the course web page for details.

```
read(a, b, c, d, e, f);
c = 1;
if(a < 0) goto L1;
a = c + 2;
d = a + b;
b = d - 3;
if(d < 0) goto L2;
goto L3;
L1:
if(c >= 10) goto L2;
e = a + b;
Z[i] = e;
c = c + 1;
goto L1;
L2:
f = a + b;
write(f);
L3:
```

(i) Identify the basic blocks, and draw a basic block graph.

(ii) Determine which basic blocks are **transparent** for the expression `a + b`, and in which basic blocks the expression is **locally available** and **locally anticipable**.

(iii) At the beginning and end of each basic block, determine whether the expression `a + b` is (globally) **available** and (globally) **anticipable**.

(iv) Perform the **earliest placement** computation. For each edge in the basic block graph, state whether the edge is the earliest place in which the expression `a + b` should be computed.

(v) Perform the **latest placement** computation. For each edge in the basic block graph, state whether the expression `a + b` could be computed "later" on the edge.

(vi) Determine on which edges a computation of `a + b` should be **inserted** by PRE.

(vii) Determine from which basic blocks the first computation of `a + b` should be **deleted** by PRE.

(viii) Write the code that results after partial redundancy elimination.

References

[1] Karl-Heinz Drechsler and Manfred P. Stadel. A variation of Knoop, Rüthing, and Steffen's lazy code motion. *SIGPLAN Not.*, 28(5):29–38, 1993.

Part 2 – Register Allocation (15 marks)

In this part of the assignment, you will perform register allocation on the following program.

```
read(a, b, c, d, e, f, g);
L1:
c = a + b
d = c * b
e = c / d
f = e - d
a = e * f
b = a - f
g = g + b
if g < 10 goto L1;
write(g);
```

(i) Draw the control flow graph.

(ii) Determine the set of variables that are live before and after each instruction.

(iii) Draw the interference graph for the program.

(iv) How large is the largest clique in the interference graph?

(v) Trace two possible runs of using Briggs' register colouring algorithm to colour the interference graph with at most three registers, such that one run succeeds, and the other fails.

(vi) Is it possible to colour the interference graph with only two registers? If so, give such a colouring. If not, show why not.

Part 3 – Instruction Scheduling (15 marks)

Consider a fictional machine architecture with the following instructions and latencies between dependent instructions:

$r_i = r_j + r_k$	1 cycle
$r_i = r_j * r_k$	2 cycles
$r_i = \text{load } *(r_j)$	3 cycles

That is, the result of an addition can be used immediately in the next cycle, but using the result of a multiplication in the instruction immediately following it incurs a one-cycle delay.

Choose an instruction scheduling priority heuristic (your own, or one from class, or a combination of heuristics), and give two sample code sequences such that:

- the priority heuristic assigns **distinct** priorities to each instruction in each of your two example code sequences, and
- list scheduling the first code sequence using your heuristic gives an optimal schedule, but list scheduling the second code sequence gives a sub-optimal schedule.

(i) Clearly define your priority heuristic.

(ii) Write down the two example code sequences.

(iii) Draw a dependence graph for each sequence.

(iv) For each sequence, determine the schedule generated using your priority heuristic.

(v) For the second code sequence, determine an optimal schedule.