HOMEWORK 2

CS 322 Compiler Construction
Winter Quarter 2005

Due : Friday, March 11 at 6:00pm

1. An *extended basic block* is a maximal sequence of basic blocks $B = \{B_1,\ B_2,\ \ldots, B_n\}$ such that B_1 has multiple predecessors or is the flow-graph's entry node, and every other block has a unique predecessor in B. Several optimizations are performed on extended basic blocks instead of standard ones.
   - Come up with an efficient algorithm to identify extended basic blocks in a flow graph. Be detailed.
   - How would you perform "local" (or more correctly, super-local) and global common subexpression elimination on extended basic blocks? Be detailed.
   - Instruction scheduling tries to improve execution speed by performing "list scheduling" (reordering a sequence of no-branch instructions) and "branch scheduling" (filling a branch delay slot or covering the delay between completing a comparison and jumping according to that comparison). Is it more or less beneficial to apply instruction scheduling on extended basic blocks rather than standard basic blocks? Briefly justify.
   - Another idea is to create extended basic blocks in a "backwards" manner: the sequence ends in a basic block that has multiple successors (or is the exit) and every other basic block in the sequence has a unique successor. Are there any optimizations that might produce better results on these blocks than on the "forward" extended blocks? Give specific examples.
2. When we discussed common subexpression elimination in class, we talked about performing it locally and then globally. Is the local optimization necessary? If yes, why? If no, what could be a reason for doing it?
3. Consider the following loop:

   ```
   for i=m to n
      a = b + i
      c = a * i
      d = b + i
   endfor
   ```

   Note that each assignment to `d` except the last one is useless. As a result of this observation we decide to perform the actual `store` operation for `d` outside the loop.
   - Under what conditions, if any, is this optimization safe?
   - What effect would this optimization have on the need of registers in the loop?
   - Describe a method to identify and then move such instructions to the end of the loop they appear in.
4. Each of the following questions refers to the flow graph in figure 1.
   - Identify the back edges and corresponding natural loops.
   - Perform available expression analysis. For each block indicate the `gen` and `kill` sets and the final versions of the `in` and `out` sets.
   - Using the available expression information from the previous part, perform global common subexpression elimination.
   - Perform live variable analysis. For each block indicate the `def` and `use` sets and the final versions of the `in` and `out` sets.

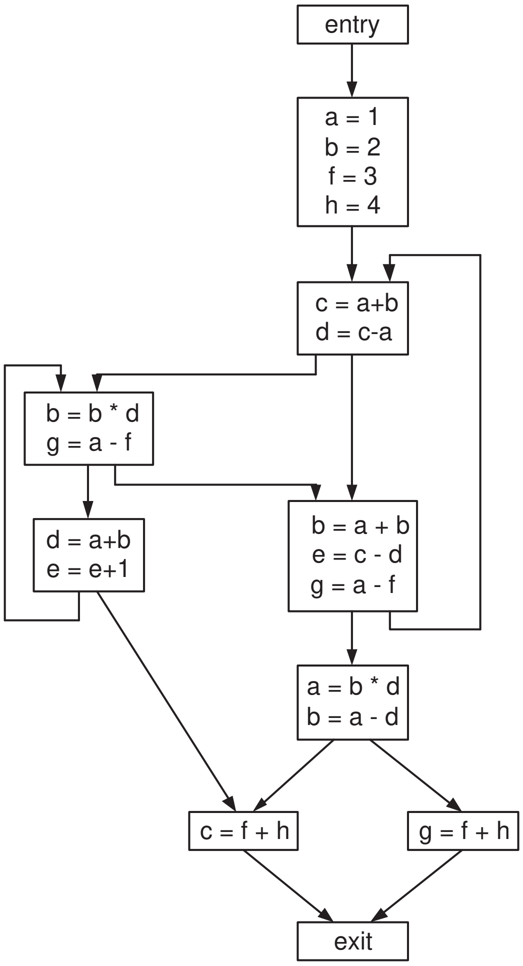

Figure 1: Flowgraph for problem 4