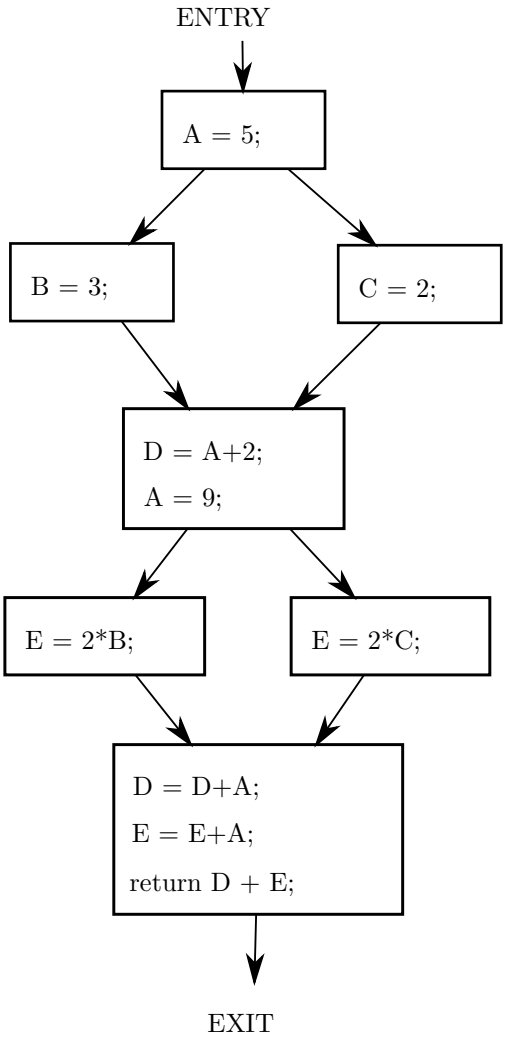
CS 243 review session, winter 2021 week 4

1. Register Allocation

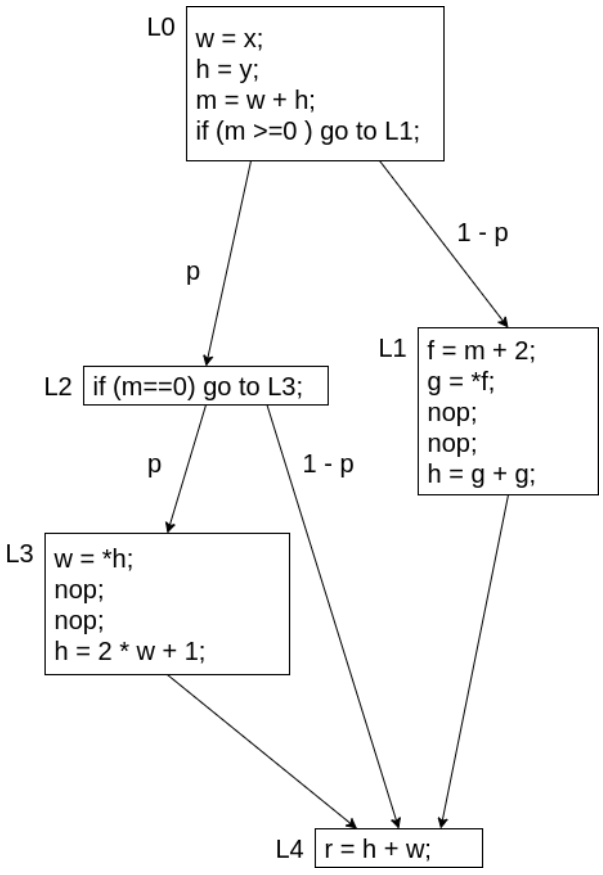
For the following control flow graph, perform register allocation. Show the results of the following steps.

1. Assign each definition and use of a variable to a live range. For example, all instances of A must be replaced with either A_1 or A_2 to signify one of two live ranges.
2. Draw the register interference graph with lines between nodes that represent live ranges.
3. Apply the heuristic-based register allocation algorithm with for a machine with 3 registers. Show the resulting "stack" of registers and show which ones, if any, are marked as spilled.
4. Assign the live ranges to registers.



2. Global instruction scheduling

Assume you have a statically scheduled machine that can only issue one operation every clock. All operations have a latency of one clock cycle, with the exception of its memory load operation, which has a latency of three clock cycles. Consider the following locally scheduled program:



Assume that only r is live at the end of the program. Each branch in the flow graph is labeled with the probability that it is taken dynamically. To answer the following, you may apply any of the code motions discussed in class, but no other optimizations.

1. Is this the best globally scheduled code that can be generated given that p = 0.1? If not, provide the improved code along with its expected execution time.

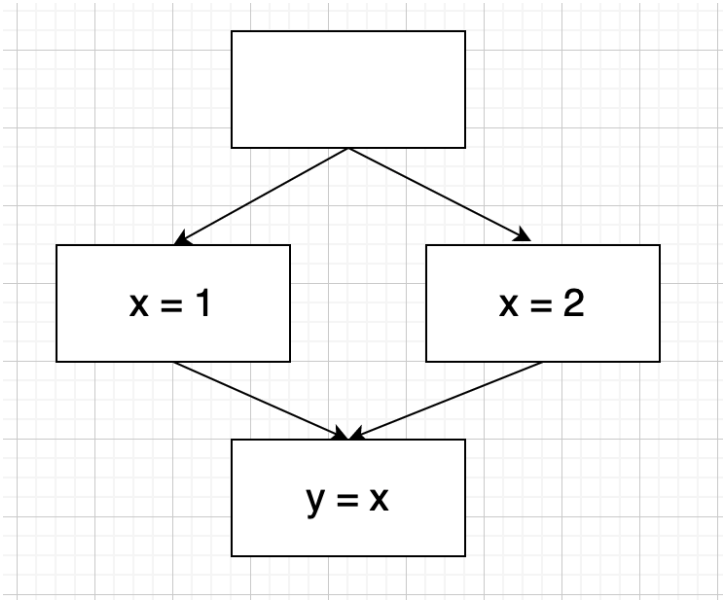
3. Enforcing data dependency in code motion

When performing global instruction scheduling on a program using code motion, we want to make sure all code motions honor existing data dependencies between instructions in the program, among other constraints.

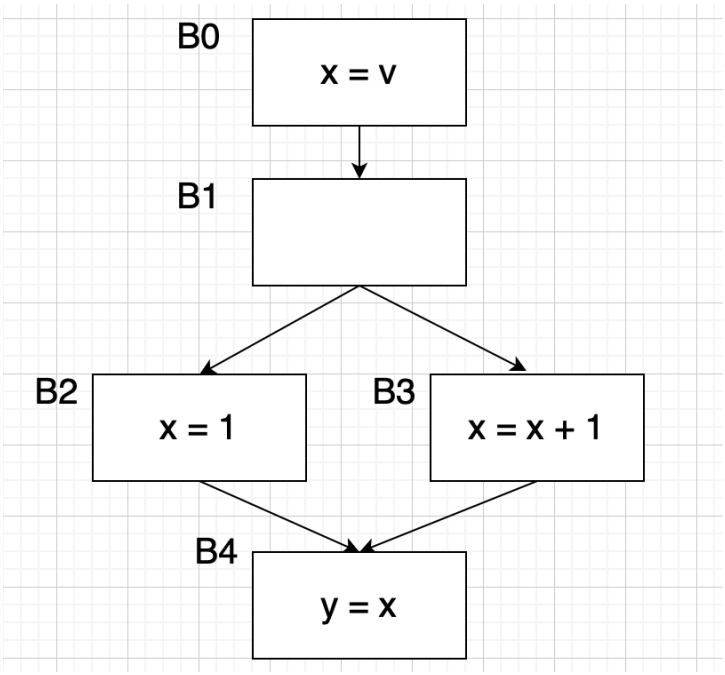
Let's look at an example. We apply the global scheduling algorithm in slide 18 of lecture 7 to the region shown in the diagram below.

Let's say we already scheduled $x = 2$ to the top block. Later on, let's assume $x = 1$ becomes the top priority instruction. Now, our algorithm would also try to schedule $x = 1$ in the top block, after $x = 2$. However, as shown in class, this violates data dependency between $x = 2$ and $y = x$ along the right branch. Obviously, our algorithm should honor this fact and reject this scheduling proposal for $x = 1$.

You task is as follows: You will help design this step in the algorithm where you check whether a given upward code motion of a definition d of the form $x = a$ or $x = a + b$ will preserve all the data dependencies. You may assume the rest of the algorithm works correctly, so that by the time d is scheduled, it is ready: all instructions it depends on have been scheduled. You should incorporate a data flow analysis to perform this step.



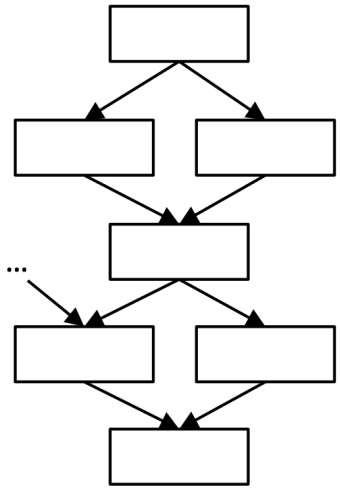
Here, your algorithm should only allow either x = 1 or x = 2 to be moved to the top block, but not both.



Here, your algorithm should allow $x = x + 1$ to be moved to B1, but not $x = 1$.

Algorithm

```
Compute data dependences;
For each region from inner to outer {
   For each basic block B in prioritized topological order {
      CandBlocks = ControlEquiv{B} ∪
                   Dominated-Successors{ControlEquiv{B}};
      CandInsts = ready operations in CandBlocks;
      For (t = 0, 1, ... until all operations from B are scheduled) {
         For (n in CandInst in priority order) {
            if (n has no resource conflicts at time t) {
               S(n) = < B, t >
               Update resource commitments
               Update data dependences
            }}
         Update CandInsts;
      }}}
```



Node *d* dominates node *n* in a graph (*d* dom *n*):

- if every path from the start node to *n* goes through *d* (a node dominates itself)

Priority functions: non-speculative before speculative