Homework Assignment #2

Due Friday, March 7, at the start of class

1. Consider the following program fragment:

```
for i := 1 to N do
 for j := 1 to N do
   s1: x[i] := j;
   s2: c[i,j+1] := x[i];
   s3: a[j,i] := (b[i,j] + c[i,j])/a[j,i];
   s4: b[i,j] := c[i,j] * 2;
 end
end
```

   a. What are the data dependences in this loop nest? Be sure to include the kind of dependence and its direction vector with each dependence.

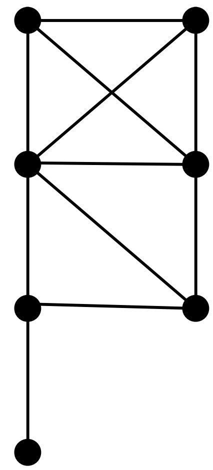
   b. Can this loop be concurrentized (compiled for a multi-processor) profitably? Why or why not? If it can be concurrentized, show how.

2. For the following procedure body, construct the interference graph, using live ranges as nodes:

```
c = 0;
for(;;) {
   a := ...;
   b := ...;
   if (...) {
      d := ...;
      f := a + d;
      break;
   }
   c := c + 1;
   if (...) {
      f := ...;
      break;
   }
   d := ...;
   b := d + 1;
   print(b);
}
print(f);
print(b);
```

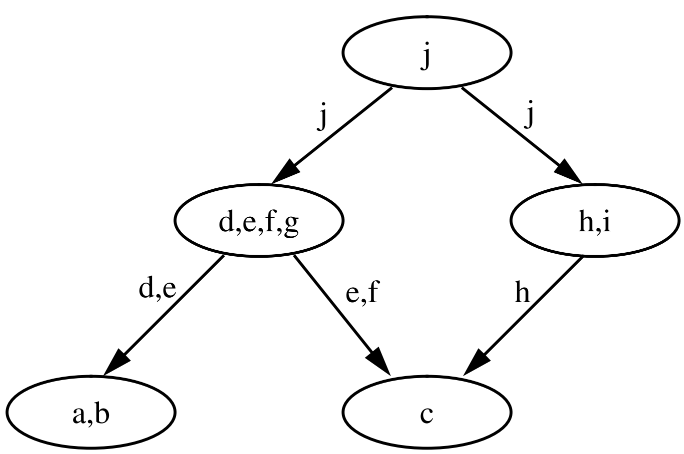
3. For the following interference graph, perform Briggs's optimistic spilling register allocation algorithm, assuming all nodes have equal weight (execution frequency), given **3** registers r1,

r2, and r3:



Show the order in which nodes are removed from the interference graph and the various allocation and spill decisions, in order.

4. Wall's link-time register allocation technique associates a list of used variables and a list of call sites with each procedure (among other things). This design assumes that all variables are live across all call sites. You are to extend this algorithm to track the set of variables live over each call site independently, in hopes of achieving better allocations.
   a. What tables would your compiler generate conveying the more precise information?
   b. How would you modify the algorithm used by the link-time allocator to exploit these tables?
   c. Show how your revised algorithm would clump variables into groups with the same register assignment for the following call graph (each node in the diagram lists the local variables used at some point in the routine, and each outgoing call arc lists the subset of the node's local variables live across that call):



   How many registers does your algorithm need to allocate all variables used in this program?

d. Show how Wall's original algorithm would clump variables into groups. How many registers does Wall's algorithm need to allocate all variables used in this program?

5. You are to develop a Gibbons & Muchnick-style code scheduler for a machine with the following kinds of interlocks:

   - cannot use the result of a **load** in the following cycle
   - cannot use the result of a **multiply** or a **divide** in the following **two** cycles

   If an interlock would occur, the compiler must insert NOP instructions to avoid violating these rules.

   a. What heuristics would you use to guide instruction selection for this machine, and in what order would you prioritize them?

   b. For the following assembly code fragment, construct and show the data dependence diagram used in the algorithm. Mark which arcs represent potential interlocks. (In the assembly code notation, the results of the operations are the third argument, and distinct variable names stand for distinct registers.)

```
s1: add i, j, k
s2: ld  [w+8], p
s3: st  k, [p+4]
s4: mul k, i, m
s5: sub a, b, t1
s6: add t1,m, t2
s7: mul m, k, t3
s8: add t2,t3,m
```

   c. Schedule this code using your heuristics. For each scheduled statement where a choice exists, identify which heuristic led you to that choice. Be sure to include NOP statements where needed.