Compilers
Fall 2007

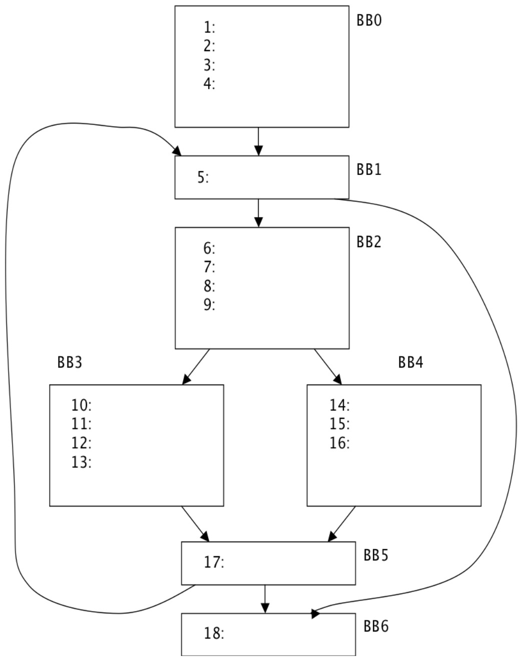
Homework 6

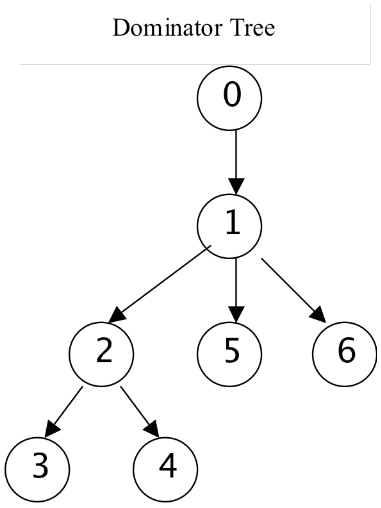
Solution

Please label all pages you turn in with your name and student number.

Problem 1: Loops and Loop Optimizations [25 points]

Answer:





1. See the CFG on the left above.
2. See the dominator trre on the right above.
3. Back edge is (5,1) since hits head dominates its tail. The natural loop is (1,2,3,4,5).
4. Could remove the statementa a = p1 to the head of the loop. This is only valies because a is not live at the end of the loop so even if the original loop were never to execute and since the head of the loop would execute once this transformation would still be correct.

Problem 2: Algebraic Transformations [15 points]

The code in red is unreachable and can thus be eliminated. The goto L1 in the end is a goto to another goto instruction and thus can be converted into a goto L0 instruction.

```
        t1 = t1 + 1
L0:     t2 = 0
        t3 = t1 * 8 + 1
        t4 = t3 + t2
        t5 = t4 * 4
        t6 = *t5
        t7 = FP + t3
       *t7 = t2
        t8 = t1
        if (t8 > 0) goto L0
L1:     goto L0
L2:     t1 = 1
        t10 = 16
        t11 = t1 * 2
        goto L1
```

after dead-code elimintation

```
        t1 = t1 + 1
L0:     t2 = 0
        t3 = (t1 << 3) + 1
        t4 = t3
        t5 = (t4 << 2)
        t6 = *t5
        t7 = FP + t3
       *t7 = t2
        t8 = t1
        if (t8 > 0) goto L0
```

after symbolic propagation for (t2 = 0) and strength reduction of the various multiplications by 4 and 8.

```
        t1 = t1 + 1
L0:     t2 = 0
        t3 = (t1 << 3) + 1
        t4 = t3
        t5 = (t3 << 2)
        t6 = *t5
        t7 = FP + t3
       *t7 = t2
        t8 = t1
        if (t1 > 0) goto L0
```

after copy propagation; now we can eliminate the variables t8 and t4 and t2 as they are never used in this code

In reality you can also see that t1 is now loop invariant and so is the sub-expression (t1 <<3) + 1 which can now be hoisted outside the loop. Along the same reasoning t5 is also loop invariant and so is everything else. This is a very uninteresting example.

Problem 3: Iterative Data Flow Analysis [60 points]

In this problem you will develop and show the application of the Live Variable analysis problem to the code of a given procedure. The live variable analysis problem seeks to determine for each scalar variable (or temporary register) if its contents is live. This has the fundamental application in the context of register allocation since if a given variable is no longer live at a given point then there is no need to keep it in register any longer.

Formally, a variable *v* is live at a execution point *p* if either its value is used at *p* or there exists an execution path from *p* to *q* along which the value of *v* is not written and is used in *q*.

a. Formalize the live variable problem as an iterative data-flow analysis problem showing the equations for the *gen* and *kill* abstractions as well as the initialization of the values for each basic block and statement. In this section you need to determine if this is a forward or backward data flow problem.

Answer:
Among the many possible formulations for this problem we can have the following:

Domain: Set of all variables
Direction: Backwards
Initial values: Empty set for all Out sets (at the output of each node since the problem flows backwards)
Function:
Gen = {v | v is used on the RHS before being redefined in the block }
Kill = { v | v is defined on the LHS of a statement in the block }

Out(b) = $U_{s\ in\ succ(b)}$ In(s), In(b) = Gen(b) U (Out(b) – Kill(b))

b. Apply your data flow problem formailization to the procedure code depicted below showing the final result of the IN and OUT set of live variables for each basic block in the code.

Answer:
See figure on the right with the live variables sets along each edge and assuming the output of BB7 has no live variables, i.e. the empty set. The table below depicts the values for the In and Out sets for each basic blocks for the various iterations of the analysis.

Iteration\BB		*BB1*	*BB2*	*BB3*	*BB4*	*BB5*	*BB6*	*BB7*
0	*In*	-	-	-	-	-	-	-
	Out	*{ }*	*{ }*	*{ }*	*{ }*	*{ }*	*{ }*	*{ }*
1	*In*	*{ }*	*{ i }*	*{ a, i }*	*{ a, b, i }*	*{ d, i }*	*{ i }*	*{ b}*
	Out	*{ i }*	*{a, b, i }*	*{a, b, d, i}*	*{ i }*	*{ i }*	*{ i }*	*{ }*
2	*In*	*{ }*	*{a, b, i }*	*{a, b, d, i}*	*{ a, b, i }*	*{ d, i }*	*{ i }*	*{ b}*
	Out	*{ a, b }*	*{a, b, d, i}*	*{a, b, d, i}*	*{ i }*	*{ i }*	*{ a, b, i }*	*{ }*
3	*In*	*{ }*	*{a, b, d, i}*	*{a, b, d, i}*	*{a, b, d, i}*	*{a, b, i}*	*{a, b, d, i}*	*{ b}*
	Out	*{ a, b, d }*	*{a, b, d, i}*	*{a, b, d, i}*	*{a, b, d, i}*	*{a, b, d, i}*	*{a, b, d, i}*	*{ }*

c. Consider that you only have 2 registers to implement the code in the body of the loop. Which set of live variables would you keep in registers and why?

Answer:
At the back edge of the loop we have 4 live variables. Given that i and a variables (that is the base address of the array a) have the longest live ranges, i.e, they are live all the time inside the body of the loop, I would leave them in registers.