

Write your answers on another sheet of paper. Homework assignments are to be completed individually. Hand written submissions are fine, but they must be readable. Due at the beginning of class. Total points: 140, 5% of course grade

1. [20 Points] Reaching expressions. An expression “ $x \text{ op } y$ ” (at statement  $s$ ) reaches a statement  $t$  if there is a path from  $s$  to  $t$  that does not go through any assignments to  $x$  or  $y$ . Write the data-flow equations for an analysis that combines available expressions with reaching expressions. In other words, both are computed at the same time. Define any notation you use.
2. [20 Points] Short Answer. What are some of the pros and cons for using the AST, 3-address code, ud/du chains, or SSA intermediate representations? Describe at least one pro and con for each.
3. [40 Points] Induction Variable Elimination. Here is an example program.

```
t = 0
loop:
  if (t>0) goto loopend
  j = j + c
  k = j*8
  y = M[k]
  t = t - y
  goto loop
loopend:
  j = j + 2
  t = t + k
  if (j<n) goto loop
```

- (a) What are the basic and derived induction variables in the example program?
- (b) Perform strength-reduction and induction-variable elimination on the example program and show the transformed program.

4. [20 Points] Alias Analysis. For each of the following code examples indicate whether the points-to or the alias pairs representation is more precise in terms of what `**p` possibly references. Explain why?

```
// code example 1
x = &y;
p = &x;
x = &z;
**p =
```

```
// code example 2
q = &z;
if ()
    p = &q;
else
    q = &y;
**p =
```

5. [40 Points] Program Optimizations. Some subset of program optimizations was applied to the code below to derive the transformed code shown below. The possible program optimizations are simple constants, copy propagation, dead-code elimination, common-subexpression elimination, PRE, and optimistic global value numbering. (If I applied global value numbering I converted the SSA back to code). Specify which optimizations were applied and in what order assuming that each optimization applied was only applied once. There is at least one other subset and/or ordering of optimizations that would result in code that does fewer arithmetic operations. Specify such a subset and its order?

```
// original code
i = 0
x = 10
if (flag) goto L1
i = 11
v = i*2
goto L2
L1:
    i = x + 1
    k = x + 1
    d = i * 2
    y = k * 2
L2:
    z = i * 2
    x = i + 2
    j = 0
loop:
    s = s + j
```

```

d = z + x
j = j + 1
if (j<=10) goto loop
print x, y, z, s, d

// optimized code
x = 10
if (flag) goto L1
i = 11
v = i*2
t1 = v
goto L2
L1:
i = x + 1
k = i
d = i * 2
y = d
t1 = d
L2:
x = i + 2
j = 0
loop:
s = s + j
d = t1 + x
j = j + 1
if (j<=10) goto loop
print x, y, t1, s, d

```