

# CMSC 430 Practice 3

These 3-address code instructions may be used in the following questions:

3-addr Instruction	Effect
load R1 x	$R1 \leftarrow x$
store x R1	$x \leftarrow R1$
add R1 R2 R3	$R1 \leftarrow R2 + R3$
sub R1 R2 R3	$R1 \leftarrow R2 - R3$
mult R1 R2 R3	$R1 \leftarrow R2 * R3$

## 1. Optimizations

- How can compiler transformations improve a program?
- What does the compiler need to consider when applying optimizations?
- What are the different scopes of compiler optimizations? What are the tradeoffs when considering what scope of optimizations to use?

## 2. Local optimizations

Consider the following code.

- $a := 1$
- $b := f + a$
- $c := a$
- $d := f + a$
- $e := f + c$
- $f := b$
- $g := f + a$

- Build a DAG for the code.

## 3. Control flow analysis

For the following problems, consider this code:

```
<S1>      a := 1
<S2>      b := 2
<S3>      L1:  c := a + b
<S4>      d := c - a
<S5>      if (...) goto L3
<S6>      L2:  d := b * d
<S7>      if (...) goto L3
<S8>      d := a + b
<S9>      e := e + 1
<S10>     goto L2
<S11>     L3:  b := a + b
<S12>     e := c - a
<S13>     if (...) goto L1
<S14>     a := b * d
<S15>     b := a - d
```

- What are the basic blocks?
- What is the control flow graph?

- Depth-first order selects nodes in the order they are visited (start by visiting the root node) and then recursively visiting every child of each node (if the child has not been visited before). Note that the order in which children are visited is random. What are all the possible results of depth-first traversal on the control flow graph?

- Using depth-first order, is it possible to visit a child before its parent? For which depth-first ordering(s) of the control flow graph does this occur?
- Postorder selects nodes (starting from root) *after* visiting every child of that node (if the child has not been visited before). Note that the order in which children are visited is random. What are all the possible results of Postorder traversal for the control flow graph?
- Reverse Postorder simply reverses the node ordering found by a Postorder traversal of the graph. What are the possible Reverse Postorder traversals of the control flow graph?
- Using Reverse Postorder, is it possible to visit a child before its parent? Why or why not?

## 4. Reaching definitions

*Reaching definitions* for a point in the program  $p$  is defined as the set of definitions of a variable for which there is some path from the definition to  $p$  with no other definition of that variable. Calculate reaching definitions for the code in the control-flow graph problem.

- What is the dataflow equation for REACH?
- In what direction is REACH calculated? I.e., does information flow forwards or backwards in the CFG?
- Calculate GEN, KILL for each basic block.
- What is a good initial value for REACH for each basic block?
- Solve the data-flow equations in reverse Postorder. Show your work.

## 5. Live variables

*Live variables* for a point in the program  $p$  is defined as the set of variables  $x$  for which there is some path from  $p$  to a use of  $x$  with no definition to  $x$  on the path. Calculate live variables for the code in the control-flow graph problem.

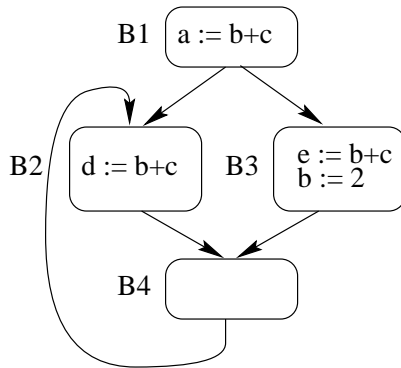
- We define  $LIVE(b)$  for a basic block  $b$  to be the set of live variables at the end of  $b$ . What is the dataflow equation for LIVE?
- In what direction is LIVE calculated? I.e., does information flow forwards or backwards in the CFG?
- Show GEN, KILL for each basic block.
- What is a good initial value for LIVE for each basic block?
- Solve the data-flow equations in rPostorder. Show your work.

---

## 6. Available expressions

*Available expressions* is a data-flow analysis problem whose solution is used to guide global common sub-expression. It calculates AVAIL, the expressions available at the beginning of each basic block.

Consider the following code. Assume that  $b+c$  is the only expression of interest:



- What is the data-flow equation for AVAIL?
- Give GEN and KILL (needed by AVAIL) for each basic block.
- What is a good initial value for AVAIL for each basic block?
- Calculate AVAIL. Show all steps, including values for AVAIL and the order basic blocks are analyzed.

---

## 7. Data-flow lattices

Prove the following properties of lattices:

- Show that  $a \leq b$  and  $b \leq c$  implies  $a \leq c$
- Show that  $a \leq (b \wedge c)$  implies  $a \leq b$

---

## 8. Data-flow frameworks

- When estimating each of the following sets, tell whether too-large or too-small estimates are conservative. Explain your answer in terms of the intended use of information.
  - Available expressions
  - Reaching definitions
  - Live variables
- What properties are necessary to ensure an iterative data-flow analysis framework terminates?
- What properties are necessary to ensure an iterative data-flow analysis framework terminates with the meet-over-all-paths solution?

---

## 9. Instruction scheduling

Consider scheduling the code below using list scheduling. All instructions must complete before executing the *jmp* instruction. Assume the following instruction latencies:

- 2-cycle latency for load
- 1-cycle latency otherwise

```
<op> <dst, s1, s2>
1  load  r1, a
2  add   r2, r1, #4
3  store x, r2
4  load  r3, b
5  mult  r4, r3, r2
6  load  r1, c
7  add   r5, r1, r3
8  store y, r5
9  load  r6, d
10 mult  r7, r5, #1
11 store z, r7
12 jmp
```

- Build the precedence graph for the instructions. Mark dependences as flow, anti, or output. You can ignore transitive dependences.
- Calculate the critical path for the instructions.
- Schedule the instructions for a single-issue processor, using forward list scheduling. Show candidates instructions at each cycle. Prioritize candidates using 1) critical path, 2) latency of instruction, 3) number of children.
- Schedule the instructions as above, for a two-issue VLIW processor.
- How could you change register assignments to improve instruction schedules in the code?