

# CMSC 430 Practice 3

## Solutions

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

- (a) How can compiler transformations improve a program?

*By reducing program size or number of instructions executed, taking advantage of redundant computations, customizing general purpose code, or undoing high-level abstractions.*

- (b) What does the compiler need to consider when applying optimizations?

*Safety, profitability, and applicability.*

- (c) What are the different scopes of compiler optimizations? What are the tradeoffs when considering what scope of optimizations to use?

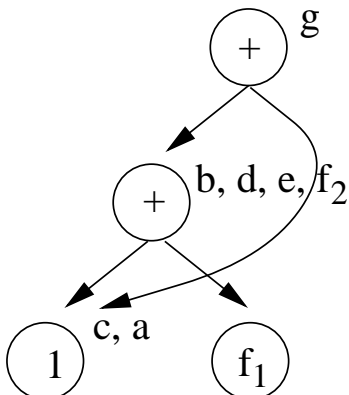
*Peephole, local, global, or interprocedural. The basic tradeoff is more complexity and compile time for optimizations with greater scope.*

### 2. Local optimizations

Consider the following code.

- (1)  $a := 1$
- (2)  $b := f + a$
- (3)  $c := a$
- (4)  $d := f + a$
- (5)  $e := f + c$
- (6)  $f := b$
- (7)  $g := f + a$

- (a) Build a DAG for the code.



*Note that assignment gives a node multiple labels, and may require renaming variables.*

### 3. Control flow analysis

For the following problems, consider this code:

```

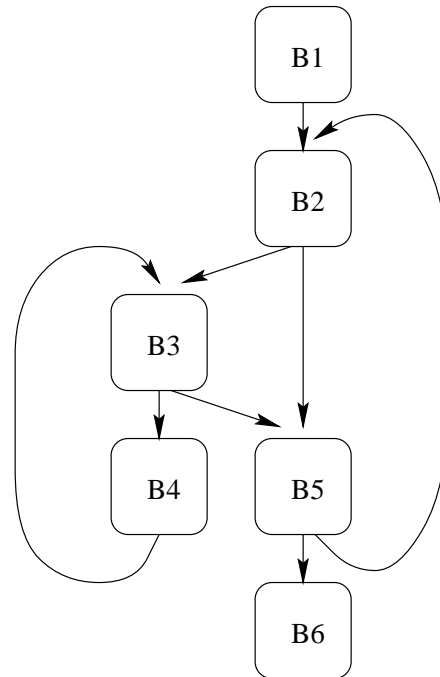
<S1>          a := 1
<S2>          b := 2
<S3>          L1: c := a + b
<S4>          d := c - a
<S5>          if (...) goto L3
<S6>          L2: d := b * d
<S7>          if (...) goto L3
<S8>          d := a + b
<S9>          e := e + 1
<S10>         goto L2
<S11>         L3: b := a + b
<S12>       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

- (a) What are the basic blocks?

```

B1 = { S1, S2 }
B2 = { S3, S4, S5 }
B3 = { S6, S7 }
B4 = { S8, S9, S10 }
B5 = { S11, S12, S13 }
B6 = { S14, S15 }
    
```

- (b) What is the control flow graph?



- (c) 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?

$B1, B2, B3, B4, B5, B6$   
 $B1, B2, B5, B6, B3, B4$   
 $B1, B2, B3, B5, B6, B4$

- (d) 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?

*If loops are present, any node ordering will cause a node to be visited before all of its parents.*

*If backward loop edges (i.e.,  $B5 \rightarrow B2, B4 \rightarrow B3$ ) are ignored, then it is still possible to visit a child before its parent using depth-first ordering. The depth-first ordering in which this occurs is:*

$B1, B2, B5, B6, B3, B4$

- (e) 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?

$B6, B5, B4, B3, B2, B1$   
 $B4, B6, B5, B3, B2, B1$

- (f) 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?

$B1, B2, B3, B4, B5, B6$   
 $B1, B2, B3, B5, B6, B4$

- (g) Using Reverse Postorder, is it possible to visit a child before its parent? Why or why not?

*If loops are present, any node ordering will cause a node to be visited before all of its parents.*

*If backward loop edges (i.e.,  $B5 \rightarrow B2, B4 \rightarrow B3$ ) are ignored, then it is not possible to visit a child before its parent using reverse postorder ordering.*

*This is because postorder will number a node only after all its children have been visited, guaranteeing a node will always be assigned a number larger than all of its children/descendants. Once the ordering is reversed, a node is guaranteed to have a smaller number than all of its children/descendants.*

#### 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.

- (a) What is the dataflow equation for REACH?

$$REACH(b) = \bigcup_{x \in pred(b)} (GEN(x) \cup (REACH(x) - KILL(x)))$$

- (b) In what direction is REACH calculated? I.e., does information flow forwards or backwards in the CFG?

*Forwards.*

- (c) Calculate GEN, KILL for each basic block.

*Note: For reaching definitions, we append to each variable definition its location to distinguish between multiple definitions to the same variable.*

Block	GEN	KILL
B1	a1,b2	a1,b2,b11,a14,b15
B2	c3,d4	c3,d4,d6,d8
B3	d6	d4,d6,d8
B4	d8,e9	d4,d6,d8,e9,e12
B5	b11,e12	b2,e9,b11,e12,b15
B6	a14,b15	a1,b2,b11,a14,b15

- (d) What is a good initial value for REACH for each basic block?

*REACH for each basic block is initialized to  $\emptyset$  (no reaching definitions).*

- (e) Solve the data-flow equations in reverse Postorder. Show your work.

*Solving the data flow equations in reverse postorder ( $B1, B2, B3, B4, B5, B6$ ), we get the following:*

Block	Data	Initial	Pass1	Pass2
B1	IN	$\emptyset$	$\emptyset$	$\emptyset$
	OUT	$\emptyset$	a1,b2	a1,b2
B2	IN	$\emptyset$	a1,b2	a1,b2,c3,d4,d6,b11,e12
	OUT	$\emptyset$	a1,b2,c3,d4	a1,b2,c3,d4,b11,e12
B3	IN	$\emptyset$	a1,b2,c3,d4	a1,b2,c3,d4,d8,e9,b11,e12
	OUT	$\emptyset$	a1,b2,c3,d6	a1,b2,c3,d6,e9,b11,e12
B4	IN	$\emptyset$	a1,b2,c3,d6	a1,b2,c3,d6,e9,b11,e12
	OUT	$\emptyset$	a1,b2,c3,d8,e9	a1,b2,c3,d8,e9,b11
B5	IN	$\emptyset$	a1,b2,c3,d4,d6	a1,b2,c3,d4,d6,e9,b11,e12
	OUT	$\emptyset$	a1,c3,d4,d6,b11,e12	a1,c3,d4,d6,b11,e12
B6	IN	$\emptyset$	a1,c3,d4,d6,b11,e12	a1,c3,d4,d6,b11,e12
	OUT	$\emptyset$	c3,d4,d6,e12,a14,b15	c3,d4,d6,e12,a14,b15

#### 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.

- (a) 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?

$$LIVE(b) = \bigcup_{x \in succ(b)} (GEN(x) \cup (LIVE(x) - KILL(x)))$$

- (b) In what direction is LIVE calculated? I.e., does information flow forwards or backwards in the CFG?

*Backwards.*

- (c) Show GEN, KILL for each basic block.

Block	GEN	KILL
B1	$\emptyset$	a,b
B2	a,b	c,d
B3	b,d	d
B4	a,b,e	d,e
B5	a,b,c	b,e
B6	b,d	a,b

- (d) What is a good initial value for LIVE for each basic block?

*LIVE for each basic block is initialized to  $\emptyset$  (no live variables).*

- (e) Solve the data-flow equations in rPostorder. Show your work.

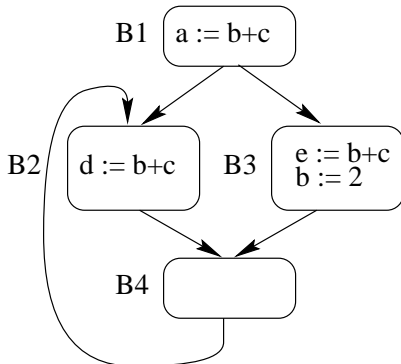
*Solving the data flow equations in postorder (B6,B5,B4,B3,B2,B1), we get the following:*

Block	Data	Initial	Pass1	Pass2
B6	OUT	$\emptyset$	$\emptyset$	$\emptyset$
	IN	$\emptyset$	b,d	b,d
B5	OUT	$\emptyset$	b,d	a,b,d,e
	IN	$\emptyset$	a,b,c	a,b,c,d
B4	OUT	$\emptyset$	$\emptyset$	a,b,c,d,e
	IN	$\emptyset$	a,b,e	a,b,c,e
B3	OUT	$\emptyset$	a,b,c,e	a,b,c,d,e
	IN	$\emptyset$	a,b,c,d,e	a,b,c,d,e
B2	OUT	$\emptyset$	a,b,c,d,e	a,b,c,d,e
	IN	$\emptyset$	a,b,e	a,b,e
B1	OUT	$\emptyset$	a,b,e	a,b,e
	IN	$\emptyset$	e	e

## 6. Available expressions

*Available expressions* is a data-flow analysis problem whose solution is used to guide global common subexpression. 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:



- (a) What is the data-flow equation for AVAIL?

$$AVAIL(b) = \bigcap_{x \in pred(b)} (GEN(x) \cup (AVAIL(x) - KILL(x)))$$

- (b) Give GEN and KILL (needed by AVAIL) for each basic block.

Block	GEN	KILL
B1	$b+c$	$\emptyset$
B2	$b+c$	$\emptyset$
B3	$\emptyset$	$b+c$
B4	$\emptyset$	$\emptyset$

- (c) What is a good initial value for AVAIL for each basic block?

*Initialize AVAIL to  $\emptyset$  (no expressions available) for all basic blocks,*

- (d) Calculate AVAIL. Show all steps, including values for AVAIL and the order basic blocks are analyzed.

Pass1

$$AVAIL(B1) = \emptyset$$

$$\begin{aligned} AVAIL(B2) &= (GEN(B1) \cup (AVAIL(B1) - KILL(B1))) \\ &\quad \cap (GEN(B4) \cup (AVAIL(B4) - KILL(B4))) \\ &= (\{b+c\} \cup (\emptyset - \emptyset)) \cap (\emptyset \cup (\emptyset - \emptyset)) \\ &= \{b+c\} \cap \emptyset = \emptyset \end{aligned}$$

$$\begin{aligned} AVAIL(B3) &= (GEN(B1) \cup (AVAIL(B1) - KILL(B1))) \\ &= (\{b+c\} \cup (\emptyset - \emptyset)) = \{b+c\} \end{aligned}$$

$$\begin{aligned} AVAIL(B4) &= (GEN(B2) \cup (AVAIL(B2) - KILL(B2))) \\ &\quad \cap (GEN(B3) \cup (AVAIL(B3) - KILL(B3))) \\ &= (\{b+c\} \cup (\emptyset - \emptyset)) \cap (\emptyset \cup (\{b+c\} - \{b+c\})) \\ &= \{b+c\} \cap \emptyset = \emptyset \end{aligned}$$

AVAIL(B3) changed, repeat

$$AVAIL(B1) = \emptyset$$

$$\begin{aligned} AVAIL(B2) &= (GEN(B1) \cup (AVAIL(B1) - KILL(B1))) \\ &\quad \cap (GEN(B4) \cup (AVAIL(B4) - KILL(B4))) \\ &= (\{b+c\} \cup (\emptyset - \emptyset)) \cap (\emptyset \cup (\emptyset - \emptyset)) \\ &= \{b+c\} \cap \emptyset = \emptyset \end{aligned}$$

$$\begin{aligned} AVAIL(B3) &= (GEN(B1) \cup (AVAIL(B1) - KILL(B1))) \\ &= (\{b+c\} \cup (\emptyset - \emptyset)) = \{b+c\} \end{aligned}$$

$$\begin{aligned} AVAIL(B4) &= (GEN(B2) \cup (AVAIL(B2) - KILL(B2))) \\ &\quad \cap (GEN(B3) \cup (AVAIL(B3) - KILL(B3))) \\ &= (\{b+c\} \cup (\emptyset - \emptyset)) \cap (\emptyset \cup (\{b+c\} - \{b+c\})) \\ &= \{b+c\} \cap \emptyset = \emptyset \end{aligned}$$

Pass2 yields no changes to AVAIL, stop

*Summary of results:*

	Init	Pass1	Pass2
AVAIL(B1)	$\emptyset$	$\emptyset$	$\emptyset$
AVAIL(B2)	$\emptyset$	$\emptyset$	$\emptyset$
AVAIL(B3)	$\emptyset$	$b+c$	$b+c$
AVAIL(B4)	$\emptyset$	$\emptyset$	$\emptyset$

---

## 7. Data-flow lattices

Prove the following properties of lattices:

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

*First, by definition of  $\leq$ , we see  $a \leq b$  gives us  $a \wedge b = a$ .*

*Similarly, from  $b \leq c$  we get  $b \wedge c = b$ .*

*Taking  $a \wedge b = a$  and replacing  $b$  with  $(b \wedge c)$ , we get  $a \wedge (b \wedge c) = a$ .*

*Since  $\wedge$  is associative, we find  $a \wedge (b \wedge c) = (a \wedge b) \wedge c = a \wedge c$ .*

*But since we also have  $a \wedge (b \wedge c) = a$ , this implies  $a \wedge c = a$ .*

*By definition of  $\leq$ , this implies  $a \leq c$ .*

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

*First, we show  $(b \wedge c) \leq b$ .*

*By definition of  $\leq$ , this is true if  $b \wedge (b \wedge c) = b \wedge c$ .*

*Since  $\wedge$  is associative, we prove it using  $b \wedge (b \wedge c) = (b \wedge b) \wedge c = b \wedge c$ .*

*Now combining  $a \leq (b \wedge c)$  and  $(b \wedge c) \leq b$  gives us  $a \leq b$ , using the solution to the previous problem.*

---

## 8. Data-flow frameworks

- (a) 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.

- i. Available expressions

*Too-small estimates are conservative ( $\perp = \emptyset$ ). Common subexpression elimination would not replace some expressions if the estimate is too small, but the answer would be still correct. In comparison, performing CSE when an expression is not available may yield incorrect results.*

- ii. Reaching definitions

*Too-large estimates are conservative ( $\perp = \{ \text{all copy statements} \}$ ). Copy propagation would be restricted if the estimate of copy statements reaching a point is too large, but the result would still be correct. In comparison, estimating too few copy statements reach a given point may enable copy propagation when the right-hand side of a missing copy statement differs from the right-hand side of the copy statements found, yielding incorrect results.*

- iii. Live variables

*Too-small estimates are conservative ( $\perp = \emptyset$ ). The opposite of variables changed by a procedure. Variables not modified by a procedure are left out of the KILL set for available expressions. Adding too few variables would result in a too-small estimate for available expressions, yielding a conservative approach to common subexpression elimination.*

- (b) What properties are necessary to ensure an iterative data-flow analysis framework terminates?

*The dataflow problem must be monotonic, I.e.,  $f(x \wedge y) \leq f(x) \wedge f(y)$ . All chains (sequences of less-than orderings) in the lattice must also have finite length (true for any lattice with finite height).*

- (c) What properties are necessary to ensure an iterative data-flow analysis framework terminates with the meet-over-all-paths solution?

*The dataflow problem must be distributive. I.e.,  $f(x \wedge y) = f(x) \wedge f(y)$ .*

## 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:

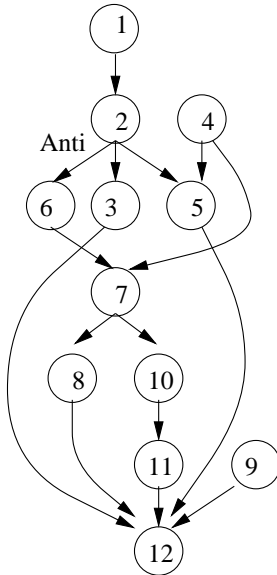
- 2-cycle latency for load
- 1-cycle latency otherwise

```

<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

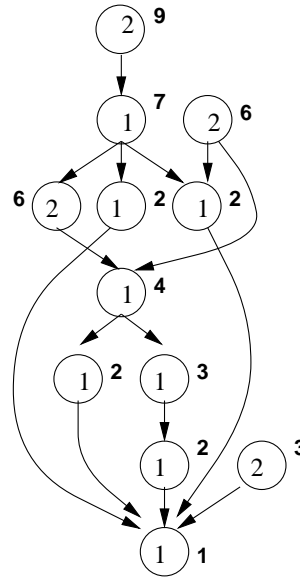
```

- (a) Build the precedence graph for the instructions. Mark dependences as flow, anti, or output. You can ignore transitive dependences.



- (b) Calculate the critical path for the instructions.

The following graph shows the critical path for a node as a number next to the node. The number inside a node indicates the latency of its instruction.



- (c) 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.

Time	1-issue		2-issue	
	Cand	Inst	Cand	Inst
1	1,4,9	1	1,4,9	1,4
2	4,9	4	9	9
3	2,9	2	2	2
4	3,5,6,9	6	3,5,6	6,3
5	3,5,9	9	5	5
6	3,5,7	7	7	7
7	3,5,8,10	10	8,10	8,10
8	3,5,8,11	11	11	11
9	3,5,8	3	12	12
10	5,8	5		
11	8	8		
12	12	12		

- (d) Schedule the instructions as above, for a two-issue VLIW processor.

See above. Results show that issuing two instructions per cycle is sufficient to achieve maximum performance (equal to critical path).

- (e) How could you change register assignments to improve instruction schedules in the code?

Using a new register instead of r1 in instruction 6 & 7 would avoid an antidependence between instruction 2 & 6.