编译原理 - 作业(5)：代码生成与优化

截至时间：2022.6.19/周一 23:59:59
提交方式：超算习堂（https://easyhpc.net/course/144）

Q1: (Exercises 6.6.1) Add rules to the syntax-directed of follow for the following control-flow constructs.

1. A repeat-statement repeat S while B.
2. A for-loop for(S1;B;S2)S3.

PRODUCTION	SEMANTIC RULES
$P \rightarrow S$	$S.next = newlabel()$ $P.code = S.code \mid\mid label(S.next)$
$S \rightarrow$ **assign**	$S.code =$ **assign**$.code$
$S \rightarrow$ **if** (B) S_1	$B.true = newlabel()$ $B.false = S_1.next = S.next$ $S.code = B.code \mid\mid label(B.true) \mid\mid S_1.code$
$S \rightarrow$ **if** (B) S_1 **else** S_2	$B.true = newlabel()$ $B.false = newlabel()$ $S_1.next = S_2.next = S.next$ $S.code = B.code$ $\mid\mid label(B.true) \mid\mid S_1.code$ $\mid\mid gen('\texttt{goto}'\ S.next)$ $\mid\mid label(B.false) \mid\mid S_2.code$
$S \rightarrow$ **while** (B) S_1	$begin = newlabel()$ $B.true = newlabel()$ $B.false = S.next$ $S_1.next = begin$ $S.code = label(begin) \mid\mid B.code$ $\mid\mid label(B.true) \mid\mid S_1.code$ $\mid\mid gen('\texttt{goto}'\ begin)$
$S \rightarrow S_1\ S_2$	$S_1.next = newlabel()$ $S_2.next = S.next$ $S.code = S_1.code \mid\mid label(S_1.next) \mid\mid S_2.code$

Figure 6.36: Syntax-directed definition for flow-of-control statements.

Production	Semantic Rule
S -> repeat S1 while B	S1.next = newlabel() B.true = newlabel()

	B.false = S.next S.code = label(B.true) || S1.code || label(S1.next) || B.code
S -> for (S1; B; S2) S3	S1.next = newlabel() B.true = newlabel() B.false = S.next S2.next = S1.next S3.next = newlabel() S.code = S1.code || lable(S1.next) || B.code || lable(B.true) || S3.code || label(S3.next) || S2.code || gen('goto', S1.next)

Q2:(p541, Exercises 8.5.1&2) For the basic block

d= b*c e= a+ b b= b*c a= e- d

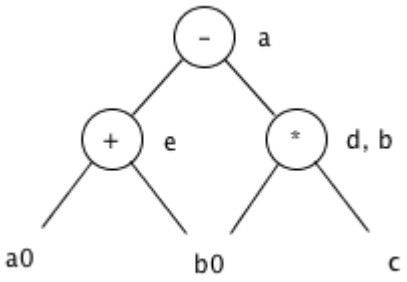
a. Construct the DAG of the block.

b. Simplify the three-address code of the block, assuming

1). Only a is live on exit from the block.

2). a, b, and c are live on exit from the block

a.



b.

1). Only a is live on exit from the block.

```
e = a + b
d = b * c
a = e - d
```

2). a, b, and c are live on exit from the block.

```
e = a + b
b = b * c
a = e - b
```

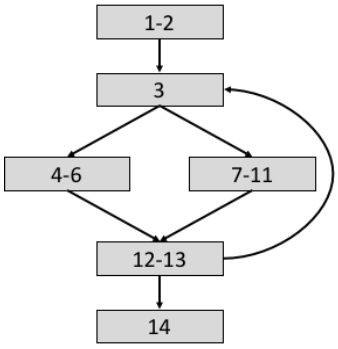
Q3: For the code segment below：

```
1:        x = 0
2:        y = 0
3:    L0: if n / 2 goto L1
4:        x = x + n
5:        y = y + 1
6:        goto L2
7:    L1: y = y + n
8:        c = 4 / 2
9:        t1 = x * c
10:       t2 = c - 1
11:       x = x + t2
12: L2: n = n -1
13:       if n > 0 goto L0
14:       return x
```

a. Partition the code segment into basic blocks.

1-2, 3, 4-6, 7-11, 12-13, 14

b. Construct the control flow graph.



c. For lines 7-11, list two optimization techniques.

t1 = x * c是dead code，可以删除

x = x + t2中t2可以通过constant folding和propagation得到，可以替换为x = x + 1

d. Suppose the whole segment is from a function '*int Func(int n)*', where *n* is the argument, *x* and *y* are local variables, then how to retrieve *n*, *x* and *y* when *Func()* is called in the final target code? Hint: consider $fp and $sp.

n: $fp + 8

x: $fp – 4

y: $fp – 8