Question 1 (6 + 4 + 6 = 16 points)

Consider the following segment of three-address code (shown in two columns to save space).

```
1:   a = 1
2:   b = 2
3:   c = a + b
4:   d = c - a
5:   if c < d goto 8
6:   d = b + d
7:   if d < 1 goto 14
8:   b = a + b
9:   e = c - a
10:  if e = 0 goto 3
11:  a = b + d
12:  b = a - d
13:  goto 17
14:  d = a + b
15:  e = e + 1
16:  goto 14
17:  return
```

1. Indicate the basic blocks of the above code segment.

   $\{1, 2\}; \{3, 4, 5\}; \{6, 7\}; \{8, 9, 10\}; \{11, 12, 13\}; \{14, 15, 16\}; \{17\}$

2. Draw the flow graph for the above segment.

   - $1, 2 \longrightarrow 3, 4, 5$
   - $3, 4, 5 \longrightarrow 6, 7$
   - $6, 7 \longrightarrow 8, 9, 10$
   - $8, 9, 10 \longrightarrow 11, 12, 13$
   - $11, 12, 13 \longrightarrow 17$
   - $3, 4, 5 \longleftrightarrow 8, 9, 10$
   - $6, 7 \longrightarrow 14, 15, 16$
   - $14, 15, 16 \longrightarrow 14, 15, 16$

3. Indicate the live variables at the end of each basic block (*not* after each statement). Assume that, initially, all variables are *not* live.

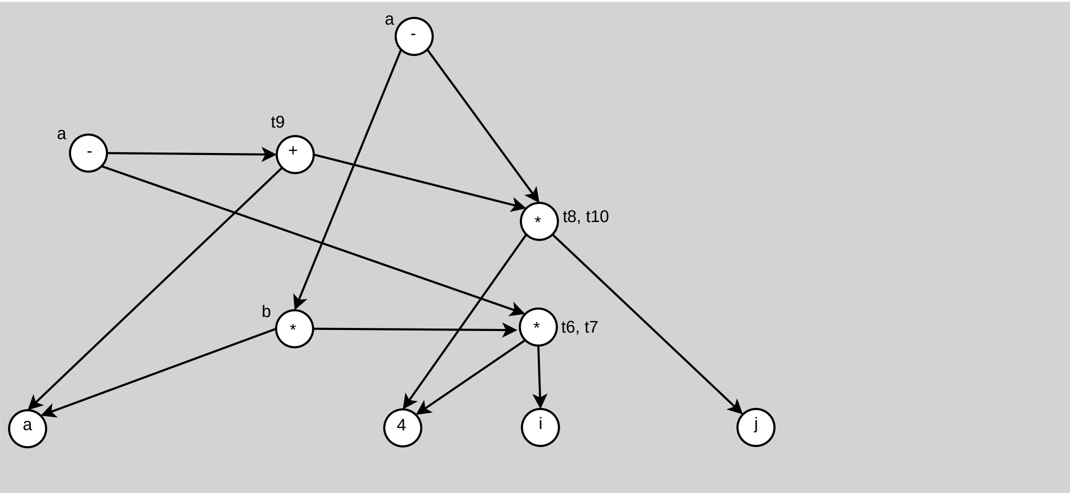
   - $1, 2 : a, b$
   - $3, 4, 5 : a, b, c, d, e$
   - $6, 7 : a, b, c, d, e$
   - $8, 9, 10 : a, b, d, e$
   - $11, 12, 13$:None
   - $14, 15, 16 : a, b, e$
   - 17 :None

Question 2 (3 + 5 = 8 points)

Consider the following segment of three-address code (shown in two columns to save space).

```
1:  t6 = 4 * i                  5:  t9 = a + t8
2:  b = a + t6                  6:  a = t7 - t9
3:  t7 = 4 * i                  7:  t10 = 4 * j
4:  t8 = 4 * j                  8:  a = b - t10
```

1. Draw the DAG representation of the above segment.



2. If only `a` is live on exit, optimize the code so that you have at most three instructions.

```
1:  t6 = i - j
2:  t7 = 4 * t6
3:  a = a - t7
```

Question 3 (14 points)

The following is an SDD for programs with simple statements and Boolean expressions.

$$
\begin{array}{llllll}
P & \longrightarrow & S & & S.next & = newlabel() \\
 & & & P.code & & = S.code \circ label(S.next) \\
 & & & & & \\
S & \longrightarrow & \mathbf{id}_1 = \mathbf{id}_2 + \mathbf{id}_3 & S.code & & = gen(\mathbf{id}_1.addr\ '='\ \mathbf{id}_2.addr\ '+'\ \mathbf{id}_3.addr) \\
 & & & & & \\
S & \longrightarrow & \mathbf{while}\ (B)\ S_1 & & & B.true = newlabel(); B.false = S.next \\
 & & & & & S_1.next = newlabel() \\
 & & & S.code & & = label(S1.next)\ \circ\ B.code \\
 & & & & & \circ\ label(B.true)\ \circ\ S_1.code \\
 & & & & & \circ\ gen('\texttt{goto}'\ S_1.next) \\
 & & & & & \\
B & \longrightarrow & B_1\ \texttt{\&\&}\ B_2 & & & B_1.true = newlabel(); B_1.false = B.false; \\
 & & & & & B_2.true = B.true; B_2.false = B.false; \\
 & & & B.code & & = B_1.code \circ label(B_1.true) \circ B_2.code \\
 & & & & & \\
B & \longrightarrow & \mathbf{id}_1 == \mathbf{id}_2 & B.code & & = gen('\texttt{if}'\ \mathbf{id}_1.addr\ '=='\ \ \mathbf{id}_2.addr\ '\texttt{goto}'\ B.true) \\
 & & & & & \circ\ \ gen('\texttt{goto}'\ B.false)
\end{array}
$$

Give the value of *P.code* as a result of parsing the string

```
while (x==y && z==u) while (x == u) x = z + y
```

Assume that generated labels are in the form `L`i, where i is an integer indicating the order in which the labels are generated; thus, the first label is `L1`, the second `L2`, and so on. (Assume top-down parsing. That is, labels generated closer to the root of the parse tree are generated earlier.)

Solution:

```
L3: if x == y goto L4
    goto L1
L4: if z == u goto L2
    goto L1
L2: L6: if x == u goto L5
        goto L3
L5: x = z + y
    goto L6
    goto L3
L1:
```

Question 4 (9 points)

The following context-free grammar generates the language $\{\mathtt{a}^n\mathtt{b}^n\mathtt{c}^* \mid n \geq 0\}$.

$$
\begin{array}{lcl}
S & \longrightarrow & T\ C \\
T & \longrightarrow & \mathtt{a}\ T\ \mathtt{b} \mid \varepsilon \\
C & \longrightarrow & \mathtt{c}\ C \mid \varepsilon
\end{array}
$$

Write an SDD for this grammar so that the start variable S has a numerical attribute *check*. For a given string, the value of $S.check$ should be zero if and only if the string is of the form $\mathtt{a}^n\mathtt{b}^n\mathtt{c}^{2^n}$ for some $n \geq 0$.

(*In constructing the SDD, make sure that the only operations performed on attributes are assignments, addition, subtraction, and multiplication.*)

$$
\begin{array}{lcll}
S & \longrightarrow & T\ C & \{S.check = T.val - C.val\} \\
T & \longrightarrow & \mathtt{a}\ T_1\ \mathtt{b} & \{T.val = 2 \times T_1.val\} \\
T & \longrightarrow & \varepsilon & \{T.val = 1\} \\
C & \longrightarrow & \mathtt{c}\ C_1 & \{C.val = C_1.val + 1\} \\
C & \longrightarrow & \varepsilon & \{C.val = 0\}
\end{array}
$$

Question 5 (6 points)

Show that an LL(1) context-free grammar is not ambiguous.

Assume that G is an LL(1) CFG which is ambiguous. Thus, there are two left-most derivations of the same string w. In particular, the two derivations have the following form:

$$S \Rightarrow^* w_1 A \gamma \Rightarrow w_1 \alpha \gamma \Rightarrow^* w_1 w_2 = w$$

and

$$S \Rightarrow^* w_1 A \gamma \Rightarrow w_1 \beta \gamma \Rightarrow^* w_1 w_2 = w$$

where $A \longrightarrow \alpha | \beta$ are rules in G. Hence, $\alpha\gamma \Rightarrow^* w_2$ and $\beta\gamma \Rightarrow^* w_2$. We have three cases:

Case 1: $\varepsilon \notin First(\alpha) \cup First(\beta)$. In this case, it must be that the first symbol of w_2 is both in $First(\alpha)$ and $First(\beta)$. This contradicts with G's being LL(1).

Case 2: $\varepsilon \in First(\alpha)$ and $\varepsilon \in First(\beta)$. Again this means that $First(\alpha) \cap First(\beta) \neq \varnothing$, which contradicts G's being LL(1).

Case 3: $\varepsilon \in First(\alpha)$ and $\varepsilon \notin First(\beta)$. In this case, we have $\gamma \Rightarrow^* w_2$. But then, the first symbol of w_2 is both in $First(\gamma)$ and $First(\beta)$. But since $First(\gamma) \subseteq Follow(A)$, this contradicts with G's being LL(1).

Question 6 (10 + 12 + 8 = 30 points)

Consider the context-free grammar $G_6 = \langle \{S, T, U\}, \{\mathtt{a}, \mathtt{b}, \mathtt{c}\}, R, S \rangle$, where R is given as follows.

$$
\begin{array}{lcl}
S & \longrightarrow & T\ \mathtt{b}\ U \\
T & \longrightarrow & \mathtt{a}\ T\ \mathtt{c} \mid \varepsilon \\
U & \longrightarrow & \mathtt{b}\ U\ \mathtt{b} \mid \mathtt{c}
\end{array}
$$

1. Draw the LALR DFA state diagram for CFG G_6.

2. Construct the LALR parsing table for G_6.

3. Trace the operation of the LR parsing algorithm on input `aaccbbcb` using the LALR parsing table from part (2) above.