1. (15 points) Construct the LR(1) parsing table for the following grammar by first computing the sets of items for each state including lookaheads for all states. Label your states according to the transition function given below and do not change the state numbers / transitions, as we will only give credit to solutions that follow this labeling.

\[ S' \rightarrow S \$
\]
\[ S \rightarrow V = E
\]
\[ S \rightarrow E
\]
\[ E \rightarrow V
\]
\[ V \rightarrow x
\]
\[ V \rightarrow * E
\]

\( N = \{S, E, V\}, T = \{=, *, x\}, \) start symbol S, augmented start symbol S'.

Start state S1: contains S' -> S$
S1 on S to S2
S1 on V to S3
S1 on E to S5
S1 on x to S8
S1 on * to S6
S3 on = to S4
S4 on E to S9
S4 on V to S7
S4 on * to S13
S4 on x to S11
S6 on V to S12
S6 on E to S10
S13 on E to S14
<table>
<thead>
<tr>
<th>x</th>
<th>*</th>
<th>=</th>
<th>$</th>
<th>S</th>
<th>E</th>
<th>V</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>6</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>7</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>8</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>9</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>10</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>11</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>12</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>13</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>14</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
2. (10 points) Given the following CFG, show what variables are live at each point. Then construct the interference graph.

\[
A = B + C \\
F = A + 3 \\
E = F + 1 \\
G = E + C \\
C = E + G \\
H = G + C
\]
3. (15 points)
   a. (6 points) Calculate the dominators of each node in this flowgraph.
   b. (4 points) Draw the dominator tree.
   c. (5 points) Identify the set of nodes in each natural loop.
4. (10 points) You are building a compiler that does inlining in a top-down fashion, i.e., when a call site is encountered during inlining, the inliner will decide whether to inline the callee procedure. Since the language you are compiling supports arbitrary recursion, you need to figure out how to prevent infinite inlining through recursive procedures.

a) What is a clean way to do this, while still allowing useful inlining? (An arbitrary cutoff after a fixed amount of inlining is not a clean way.) (5 points)

b) How can you extend your solution to allow for a limited amount of inlining for recursive procedures, akin to loop unrolling? (5 points)
5. (15 points) Consider scheduling the following basic block for a target machine where a load or multiply takes 2 cycles, interlocking with the following instructions for a cycle if that instruction uses the result of the load or multiply.

(1) $x := *p$
(2) $y := x*x$
(3) $z := x+y$
(4) $u := *q$
(5) $v := u+5$

// p, q, z, and v are live at this point

i. (2 points) Identify the interlocks (if any) in this schedule.
ii. (5 points) Show the data dependence graph for these statements.
iii. (8 points) Perform the list scheduling algorithm to construct a different (better) schedule. At each step in the algorithm, identify the candidate instructions, the best instruction selected and the reason (heuristic rule) why. Show the current clock time in the algorithm at each step.
6. (15 points) Consider the calculation of reaching definitions on the following program:

\[
\begin{align*}
    x &:= 1; \\
    y &:= 1; \\
    \text{if } (z \neq 0) \text{ then } x := 2 \text{ else } y := 2; \\
    w &:= x+y
\end{align*}
\]

a. (2 points) Draw a CFG for this program.
b. (1 point) Show the topological order traversal for this CFG.
c. (5 points) Calculate reaching definitions, showing the result of each iteration. How many iterations are required?
d. (7 points) Prove that when reaching definitions is computed by iteration on an acyclic graph, taking the nodes in topological order only one iteration is necessary. Hint: Prove and make use of the lemma that each node is visited after all its predecessors.
7. (25 points) Convert the following program into SSA form. Show your work after each stage.
   a. (1 point) Add a start node containing initializations of all variables
   b. (2 points) Draw the dominator tree
   c. (8 points) Calculate the dominance frontiers for all nodes
   d. (8 points) Calculate the iterated dominance frontiers for those nodes that require them.
   e. (2 points) Insert $\phi$-functions
   f. (2 points) Add subscripts to variables
   g. (2 points) Convert back from SSA form by inserting move-instructions in place of $\phi$-functions.