Reminder: Assignments are individual assignments, not done in pairs. The work must be all your own.

You may hand in a handwritten solution or a printout of a typeset solution at the beginning of lecture on Tuesday, September 9. Please read the late policy for written assignments on the course web page. If you decide not to typeset your answers, make sure the text and pictures are legible and clear.

Problem 1 (20 points)

(a) Construct an abstract syntax tree (AST) corresponding to the following expressions. For example, the AST for the expression

\{ 1 - 2 + 3; \}

would be

PLUS(MINUS(CONST(1),CONST(2)),CONST(3))

Here we use a term notation for trees where the node is the constructor and the subtrees are arguments. Use the names PLUS, MINUS, TIMES, and DIV for the constructors.

(i) \{ (1 + 2) *3; \}
(ii) \{ 7 - (1 - 2 *3) ); \}
(iii) \{ 7 + 10 + (1/2 *3); \}

(b) Translate the AST from problem 1(a)(c) into linear three-address form by applying maximal munch using the patterns in the table below. Subtrees should be translated left-to-right. Temporaries should be called t0, t1, ..., tn. A wildcard "_" in the pattern matches an arbitrary subtree.

<table>
<thead>
<tr>
<th>Pattern</th>
<th>IR</th>
</tr>
</thead>
<tbody>
<tr>
<td>CONST(c)</td>
<td>ti &lt;- c</td>
</tr>
<tr>
<td>DIV(_,CONST(c))</td>
<td>tj &lt;- ti / c</td>
</tr>
<tr>
<td>DIV(<em>,</em>)</td>
<td>tj &lt;- ti / tk</td>
</tr>
<tr>
<td>PLUS(<em>,</em>)</td>
<td>ti &lt;- tj + tk</td>
</tr>
<tr>
<td>MINUS(<em>,</em>)</td>
<td>ti &lt;- tj - tk</td>
</tr>
<tr>
<td>TIMES(<em>,</em>)</td>
<td>ti &lt;- tj * tk</td>
</tr>
</tbody>
</table>
For example, the AST from part (a) would be translated to

\[
\begin{align*}
t_0 & \leftarrow 1 \\
t_1 & \leftarrow 2 \\
t_2 & \leftarrow t_0 - t_1 \\
t_3 & \leftarrow 3 \\
t_4 & \leftarrow t_2 + t_3
\end{align*}
\]

**Problem 2 (20 points)**

(a) Compute the live variables after each statement in the following programs (assuming some more complex instructions than before):

(i) \[
\begin{align*}
t_1 & \leftarrow 2 \\
t_2 & \leftarrow 3 \\
a & \leftarrow t_1 + t_2 \\
t_3 & \leftarrow a \\
t_4 & \leftarrow t_2 - t_1 \\
t_5 & \leftarrow t_4 + a + t_3
\end{align*}
\]

(ii) \[
\begin{align*}
t_1 & \leftarrow 2 \\
b & \leftarrow 5 \\
t_2 & \leftarrow 3 \\
a & \leftarrow 3 + b \\
a & \leftarrow a + t_2 \\
t_3 & \leftarrow a \\
a & \leftarrow 2 * a \\
b & \leftarrow a * 3 \\
t_4 & \leftarrow b + t_1 + t_3
\end{align*}
\]

(b) Construct the interference graph for both programs. Are they chordal? Justify your answers\(^1\)

(c) Consider the following program

\[
\begin{align*}
t_1 & \leftarrow 3 \\
t_2 & \leftarrow 2 \\
t_3 & \leftarrow 1 \\
t_4 & \leftarrow t_3 + 1 \\
t_5 & \leftarrow t_2 * 3 \\
t_6 & \leftarrow t_1 - 10
\end{align*}
\]

(i) Use the algorithm from Lecture 3 to allocate registers for the program above.

(ii) Does your allocation use the minimum number of registers for this interference graph?

(iii) Is it possible to reorder the statements so that the program gives the same result, but there is less interference (i.e., fewer registers are needed)? How many register do we need now?

---

\(^1\)Refer to the lecture notes for Lecture 3 at http://www.cs.cmu.edu/~fp/courses/15411-f09/lectures/03-regalloc.pdf.
Problem 3 (20 points)

For this section, we are going to use the following assembly language:

<table>
<thead>
<tr>
<th>Instruction</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ADD r1 r2 r3</td>
<td>r3 ← r1 + r2</td>
</tr>
<tr>
<td>SUB r1 r2 r3</td>
<td>r3 ← r1 − r2</td>
</tr>
<tr>
<td>MUL r1 r2 r3</td>
<td>r3 ← r1 * r2</td>
</tr>
<tr>
<td>DIV r1 r2 r3</td>
<td>r3 ← r1 / r2</td>
</tr>
<tr>
<td>MOVE r1 r2 r3</td>
<td>r2 ← r1</td>
</tr>
<tr>
<td>LOAD r1 c</td>
<td>r1 ← c, where c is a constant</td>
</tr>
<tr>
<td>Label:</td>
<td>identifies a program point</td>
</tr>
<tr>
<td>JUMP label</td>
<td>The execution of the program jumps to label</td>
</tr>
<tr>
<td>JUMPZERO ri label</td>
<td>If r0 is zero, the execution jumps to label. Otherwise, it resumes right after the JUMPZERO instruction</td>
</tr>
</tbody>
</table>

(a) Translate the following program into assembly, selecting instructions and allocating the registers.

```
a = 3;
b = 5;
return (a+b)*(a-b)*3;
```

(b) Using Jump and JumpZero, handwrite assembly code for a = e1 && e2 and a = e1 || e2, where e1 and e2 are expressions. Assume that the variable a has been assigned register ra, that E1 and E2 stand for the assembly code corresponding to e1 and e2 respectively, and that both these expressions store their results in register re.

<table>
<thead>
<tr>
<th>e1 &amp;&amp; e2</th>
<th>If e1 is FALSE, then the result is FALSE (e2 is not evaluated). Otherwise, the result is the result of e2</th>
</tr>
</thead>
<tbody>
<tr>
<td>e1</td>
<td></td>
</tr>
<tr>
<td>FALSE</td>
<td>The constant 0</td>
</tr>
<tr>
<td>TRUE</td>
<td>Any non-zero value</td>
</tr>
</tbody>
</table>