1. Since we want to use as few registers as possible, the instructions will look like:

\[ op \; R_i, \; R_i, \; R_j \]

First, we must modify the labeling algorithm. It cannot be the same as before because now we keep all operands in registers and we need a \textit{load} for each one (therefore, all leaves should have label 1).

\begin{verbatim}
if \( n \) is a leaf then \( label(n) := 1 \)
else begin /* \( n \) is an interior node */
  let \( n_1, n_2 \) be \( n \)'s left and right child respectively
  if \( label(n_1) = label(n_2) \) then \( label(n) = label(n_1) + 1 \)
  else \( label(n) = \max\{label(n_1), \; label(n_2)\} \)
end
\end{verbatim}

The labeled tree will be:

```
    t4 3
   / \
 t1 2 / \
 a 1 b 1
```

```
 t3 2
 / \
 e 1
```

```
 t2 2
/ \
 c 1 d 1
```
procedure gencode(n);
begin
  if $n$ is a leaf representing operand $\text{name}$ then
    print 'MOV' $\|$ $\text{name}$ $\|$ 'r' $\|$ $\text{top(rstack)}$;
  else if $n$ is an interior node with left child $n_1$ and right child $n_2$ then
    if $1 \leq \text{label}(n_1) \leq \text{label}(n_2)$ and $\text{label}(n_1) < r$ then begin
      swap(rstack);
      gencode(n_2);
      $R := \text{pop(rstack)}$;
      gencode(n_1);
      print $\text{op}$ $\|$ $\text{top(rstack)}$ $\|$ 'r' $\|$ $\text{top(rstack)}$ $\|$ $R$;
      push(rstack, $R$);
      swap(rstack);
    end
    else if $1 \leq \text{label}(n_2) \leq \text{label}(n_1)$ and $\text{label}(n_2) < r$ then begin
      gencode(n_1);
      $R := \text{pop(rstack)}$;
      gencode(n_2);
      print $\text{op}$ $\|$ $R$ $\|$ 'r' $\|$ $R$ $\|$ 'r' $\|$ $\text{top(rstack)}$;
      push(rstack, $R$);
    end
    else begin /* both labels geq r */
      gencode(n_2);
      $T := \text{pop(tstack)}$;
      print 'MOV' $\|$ $\text{top(rstack)}$ $\|$ 'r' $\|$ $T$;
      gencode(n_1);
      $R := \text{pop(rstack)}$;
      print 'MOV' $\|$ $T$ $\|$ 'r' $\|$ $\text{top(rstack)}$;
      print $\text{op}$ $\|$ $R$ $\|$ 'r' $\|$ $R$ $\|$ 'r' $\|$ $\text{top(rstack)}$;
      push(rstack, $R$);
      push(tstack, $T$);
    end
end

The code generated for the labeled tree is shown on the next page. Also shown are the sequence of calls to $\text{gencode}$ and the contents of $\text{rstack}$ at the time of each call (in brackets, top at left end).
\[
\text{gen code}(t_1) \quad [R_0 R_1 R_2] \\
\text{gen code}(t_1) \quad [R_0 R_1 R_2] \\
\text{gen code}(a) \quad [R_0 R_1 R_2] \\
\text{MOV} \; a, \; R_0 \\
\text{gen code}(b) \quad [R_1 R_2] \\
\text{MOV} \; b, \; R_1 \\
\text{ADD} \; R_0, \; R_0, \; R_1 \\
\text{gen code}(t_3) \quad [R_1 R_2] \\
\text{gen code}(t_2) \quad [R_2 R_1] \\
\text{gen code}(c) \quad [R_2 R_1] \\
\text{MOV} \; c, \; R_2 \\
\text{gen code}(d) \quad [R_1] \\
\text{MOV} \; d, \; R_1 \\
\text{ADD} \; R_2, \; R_2, \; R_1 \\
\text{gen code}(e) \quad [R_1] \\
\text{MOV} \; e, \; R_1 \\
\text{SUB} \; R_1, \; R_1, \; R_2 \\
\text{SUB} \; R_0, \; R_0, \; R_1
\]

2.

3. Let’s number the statements:
   (1) \(d := b \times c\)
   (2) \(e := a + b\)
   (3) \(b := b \times c\)
   (4) \(a := e - d\)

   (a) Since \(a\) is live at the end, (1) and (2) have to go before (4).
   Similarly, (3) must go after (1) and (2).
   Therefore, the valid evaluation orders are:
   (1)(2)(3)(4)
   (1)(2)(4)(3)
   (2)(1)(3)(4)
   (2)(1)(4)(3)

   Another solution suggested by one student says that since \(d\) is not alive, one can eliminate (1) all together and replace \(d\) by \(b\) (since they compute the same value) in (4), call
it (4'). This also gives us one legal evaluation order:

(2)(3)(4')

(b) Since (a) is live at the end, (1) and (2) have to go before (4).
Statement (3) can be dropped.
Therefore, the valid evaluation orders are:
(1)(2)(4)
(2)(1)(4)