Consider this simple translation of our favorite Fibonacci function:

```
fib:  a := r3
     f := r1
     n2 := M[f]
     n1 := M[f + 4]
     x := M[f + 8]
     n0 := n1 + n2
     M[f + 4] := n0
     M[f] := n1
     if x <= 3 goto L1
     x := x - 1
     M[f + 8] := x
     r1 := f
     call fib
     n0 := r1
L1:   r1 := n0
     r3 := a
     return
```

The code has been compiled for a target architecture with four (4) registers:

- $r1$ is a (caller-save) argument/result register,
- $r2$ and $r4$ are caller-save temporaries.
- $r3$ is a callee-save register.

The code defines and uses temporaries $n0$, $n1$, $n2$, $a$, $x$, and $f$. 
1. Build the control flow graph for this program fragment, and perform liveness analysis on it. Show the in and out sets for each node in the graph. (There should be one node for each line of code.)

<table>
<thead>
<tr>
<th>n</th>
<th>dce(n)</th>
<th>use(n)</th>
<th>in(n)</th>
<th>out(n)</th>
</tr>
</thead>
<tbody>
<tr>
<td>fib:</td>
<td>a := r3</td>
<td>{a}</td>
<td>{r3}</td>
<td>{r1,r3}</td>
</tr>
<tr>
<td></td>
<td>f := r1</td>
<td>{f}</td>
<td>{r1}</td>
<td>{r1,a}</td>
</tr>
<tr>
<td></td>
<td>n2 := M[f]</td>
<td>{n2}</td>
<td>{f}</td>
<td>{a,f}</td>
</tr>
<tr>
<td></td>
<td>n1 := M[f + 4]</td>
<td>{n1}</td>
<td>{f}</td>
<td>{n2,a,f}</td>
</tr>
<tr>
<td></td>
<td>n0 := n1 + n2</td>
<td>{n0}</td>
<td>{n1,n2}</td>
<td>{n1,n2,a,f, x}</td>
</tr>
<tr>
<td></td>
<td>M[f + 4] := n0</td>
<td>{}</td>
<td>{n0,f}</td>
<td>{n0,n1,a,f, x}</td>
</tr>
<tr>
<td></td>
<td>M[f] := n1</td>
<td>{}</td>
<td>{n1,f}</td>
<td>{n0,n1,a,f, x}</td>
</tr>
<tr>
<td>if x &lt;= 3 g...</td>
<td>{}</td>
<td>{x}</td>
<td>{a,f,x}</td>
<td>{a,f}</td>
</tr>
<tr>
<td></td>
<td>x := x - 1</td>
<td>{x}</td>
<td>{x}</td>
<td>{a,f,x}</td>
</tr>
<tr>
<td></td>
<td>M[f + 8] := x</td>
<td>{}</td>
<td>{f,x}</td>
<td>{a,f,x}</td>
</tr>
<tr>
<td></td>
<td>r1 := f</td>
<td>{r1}</td>
<td>{f}</td>
<td>{a,f}</td>
</tr>
<tr>
<td>call fib</td>
<td>r1 := n0</td>
<td>{r1}</td>
<td>{r1}</td>
<td>{r1,a}</td>
</tr>
<tr>
<td></td>
<td>n0 := n1</td>
<td>{n0}</td>
<td>{r1}</td>
<td>{r1,a}</td>
</tr>
<tr>
<td></td>
<td>r3 := a</td>
<td>{r3}</td>
<td>{a}</td>
<td>{r1,a}</td>
</tr>
<tr>
<td>return</td>
<td>{}</td>
<td>{r1,r3}</td>
<td>{r1,r3}</td>
<td>{r1,r3}</td>
</tr>
</tbody>
</table>

2. Construct the interference graph for the program fragment, using solid lines to show interference, and dotted lines to show MOVE relations.
3. Show the steps for register allocation (graph coloring with conservative coalescing) for this program in detail, as described in the notes and pages 229–232 of the textbook. When choosing nodes for potential spills, use the spill priority heuristic \((\text{number of def's + number of uses}) / \text{degree}\). Use the George criteria for conservative coalescing.

- No simplifies possible. (All non-move-related nodes significant.)
- No coalesces possible. (All move-related pairs fail conservative criteria.)
- No freezes possible. (No move-related nodes of low degree.)
- Must choose potential spill. Use heuristic:

<table>
<thead>
<tr>
<th>Node</th>
<th>def's + uses</th>
<th>degree</th>
<th>spill priority</th>
</tr>
</thead>
<tbody>
<tr>
<td>n0</td>
<td>4</td>
<td>4</td>
<td>1</td>
</tr>
<tr>
<td>n1</td>
<td>3</td>
<td>5</td>
<td>0.6</td>
</tr>
<tr>
<td>n2</td>
<td>2</td>
<td>4</td>
<td>0.5</td>
</tr>
<tr>
<td>a</td>
<td>2</td>
<td>6</td>
<td>0.33</td>
</tr>
<tr>
<td>f</td>
<td>8</td>
<td>5</td>
<td>1.6</td>
</tr>
<tr>
<td>x</td>
<td>5</td>
<td>5</td>
<td>1</td>
</tr>
</tbody>
</table>

Node a wins. Push a on stack as potential spill.

- Simplify n2.
- Simplify n1.
- Simplify x.

- Can’t simplify – all nodes are either move-related or precolored.

Can coalesce r1, f or r1, n0. Last move constrained. Eliminate move, simplify n0.
• Select phase reveals no coloring for node $a$. Becomes actual spill. Add code into original program fragment for spilling:

$$a := r3 \quad \rightarrow \quad a1 := r3$$

$$M[a_{\text{loc}}] := a1$$

$$r3 := a \quad \rightarrow \quad a2 := M[a_{\text{loc}}]$$

$$r3 := a2$$

• Recalculate liveness ranges, build new interference graph:

- Simplify $n_2$.
- Simplify $n_1$.
- Simplify $x$.

• Coalesce $a_1$, $a_2$, and $r_3$.
• Coalesce $f$, $r_1$.
• Simplify $n_0$.

• Select colors:

<table>
<thead>
<tr>
<th></th>
<th>$r_1$</th>
<th>$r_2$</th>
<th>$r_3$</th>
<th>$r_4$</th>
<th>$n_0$</th>
<th>$n_1$</th>
<th>$n_2$</th>
<th>$a_1$</th>
<th>$a_2$</th>
<th>$f$</th>
<th>$x$</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>2</td>
<td>4</td>
<td>2</td>
<td>3</td>
<td>3</td>
<td>1</td>
<td>3</td>
</tr>
</tbody>
</table>
4. Show the final program after register allocation,

```plaintext
fib:  r3 := r3
     M[a_loc] := r3
     r1 := r1
     r2 := M[r1]
     r4 := M[r1 + 4]
     r3 := M[r1 + 8]
     r2 := r4 + r2
     M[r1 + 4] := r2
     M[r1] := r4
     if r3 <= 3 goto L1
     r3 := r3 - 1
     M[r1 + 8] := r3
     r1 := r1
     call fib
     r2 := r1
L1:  r1 := r2
     r3 := M[a_loc]
     r3 := r3
     return
```

... with redundant moves removed.

```plaintext
fib:  M[a_loc] := r3
     r2 := M[r1]
     r4 := M[r1 + 4]
     r3 := M[r1 + 8]
     r2 := r4 + r2
     M[r1 + 4] := r2
     M[r1] := r4
     if r3 <= 3 goto L1
     r3 := r3 - 1
     M[r1 + 8] := r3
     call fib
     r2 := r1
L1:  r1 := r2
     r3 := M[a_loc]
     return
```