Register Allocation Example

In the previous unit we saw how register allocation can be accomplished via a graph coloring technique. In this section we are going to extend the graph coloring to the entire program. Previously we saw the application of graph coloring to a basic block. In this example we will see graph coloring extended to a control flow graph representing an entire program. To simplify the example, we will be only looking at a program segment, one that includes a do-until-loop and an if-then-else construct.

Suppose that we have the following program segment:

```plaintext
DO
    a ← b + c;
    d ← - a;
    e ← d + f;
    IF _____ THEN
        f ← 2 * e;
    ELSE
        b ← d + e;
        e ← e - 1;
    END
    b ← f + c;
UNTIL _____ END
```

The control flow graph for the above program segment is
Where basic block 1 contains

\[
\begin{align*}
a & \leftarrow b + c; \\
d & \leftarrow -a; \\
e & \leftarrow d + f;
\end{align*}
\]

block 2 contains

\[
f \leftarrow 2 \times e;
\]

block 3 contains

\[
\begin{align*}
b & \leftarrow d + e; \\
e & \leftarrow e - 1;
\end{align*}
\]

and block 4 contains

\[
b \leftarrow f + c;
\]

This is depicted in the annotated control flow graph below:
In the example we will assume that $b$ is live after the code segment is executed.

The next step in our allocation approach is to determine which variables are live at each point in the program segment. This is called live range analysis. Initially, we will assume that only $b$ is live on the exit from the program segment. The graph below shows which variables are live at each point in the program segment.
Ideally, we want to assign a variable to a register over its entire live time, from the instant that it is born to its last usage. We will again represent the constraints on non-interfering live ranges via a register interference graph. The nodes in the graph are variable names. There is an undirected edge between two nodes if those nodes represent variables that are live at the same time. Intuitively, an edge between two variables says that these variables must be in different registers. The register interference graph for the above example is:
If we try a 3-coloring, we see that we can cut the node a since its degree is 2. However, if we try any other cuts, we see that all remaining nodes have a degree of 3 or greater. Continuing with Preston’s approach of continuing to cut where the degree is equal to $n$, we get the following stack of nodes:

```
e
b
f
d
c
a
```

3-coloring the interference graph below we get

We color node e red, node b green, node f blue, node d green, and then we try to color node c and discover that we cannot color it red, green, or blue. Hence, we do not have a 3-color for this interference graph. We next try a 4-color.
We can use our same approach to cutting nodes that we did above. From the same stack of cut nodes we have

```
e
d
c
b
f
```

and the coloring that we get is

```
We repeating the coloring we have
```

```
Hence, we see that if we assign a and b to register R1, f and c to R2, d to R3, and e to R4, we can keep those registers in the same registers over the lifetime of the program segment. This assumes that there are sufficient registers for the intermediate results. When that assumption is not correct, then we will need to spill registers. That topic is covered in another unit.

Note that the only difference from what we have done before is the calculation of the live ranges.