Register Allocation

• **Goal:** replace temporary variable accesses by register accesses
• **Why?**
• **Constraints:**
  – Fixed set of registers
  – Two simultaneously live variables cannot be allocated to the same register

A variable is **live** if it will be used again before being redefined.
1. Identify Live Variable Ranges

Basic rule:
Temporaries $t_1$ and $t_2$ can share the same register if at any point in the program at most one of $t_1$ or $t_2$ is live!

- Compute live variables for each point:
Register Interference Graph

- We construct an undirected graph
  - A node for each temporary
  - An edge between $t_1$ and $t_2$ if they are live simultaneously at some point in the program

- This is the register interference graph (RIG)
  - Two temporaries can be allocated to the same register if there is no edge connecting them
What is the Register Inference Graph for this example?

- Compute live variables for each point:

  \[
  \begin{align*}
  a & := b + c \\
  d & := -a \\
  e & := d + f \\
  f & := 2 \times e \\
  b & := f + c \\
  \end{align*}
  \]
1. What **cannot** be assigned same register?
2. What **can** be assigned the same register?
Your Turn – Write down the live variables after each statement. Hint: Start at the bottom.

Instructions       Live vars

b = a + 2

c = b * b

b = c + 1

return b * a
Live Variables

Instructions

b = a + 2

c = b * b

b = c + 1

b,a

return b * a
Instructions

Live vars

\[ b = a + 2 \]

\[ c = b \times b \]

\[ b = c + 1 \]

return \( b \times a \)
Instructions | Live vars
---|---
b = a + 2 | b,a
\( c = b \times b \) | a,c
\( b = c + 1 \) | b,a
\text{return } b \times a
Instructions

b = a + 2

c = b * b

b = c + 1

return b * a

Live vars

a

b,a

a,c

b,a

b,a
Interference graph and Register Allocation

- **Nodes** of the graph = variables
- **Edges** connect variables that interfere with one another
- Nodes will be assigned a color corresponding to the register assigned to the variable
- Two colors can’t be next to one another in the graph
Register Allocation = Graph Coloring

• A **coloring of a graph** is an assignment of colors to nodes, such that nodes connected by an edge have different colors.

• A graph is **k-colorable** if it has a coloring with \( k \) colors.
Coloring the RIG

Instructions

\[
b = a + 2
\]

\[
c = b \times b
\]

\[
b = c + 1
\]

\[
\text{return } b \times a
\]

Live vars

\[
a
\]

\[
a, b
\]

\[
a, c
\]

\[
a, b
\]

<table>
<thead>
<tr>
<th>color</th>
<th>register</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>R1</td>
</tr>
<tr>
<td></td>
<td>R2</td>
</tr>
</tbody>
</table>

Diagram:

```
  a
 /|
/ |
 b
```

```
  a
 /|
/ |
 c
```

a, b, c

Coloring the RIG

Instructions

\begin{align*}
    b &= a + 2 \\
    c &= b \times b \\
    b &= c + 1 \\
    \text{return } b \times a
\end{align*}

Live vars

- a
- a,b
- a,c
- a,b

Live vars

- a
- a,b
- a,c
- a,b

Coloring

- Color
- Register

\begin{align*}
    \text{color} & \quad \text{register} \\
    \text{R1} & \quad \text{R2}
\end{align*}
How to do the Graph coloring

• Questions:
  – Can we efficiently find a coloring of the graph whenever possible?
  – Can we efficiently find the **optimum coloring** of the graph?
  – How do we choose registers to avoid move instructions?
  – What do we do when there aren’t enough colors (registers) to color the graph?
• Kempe’s algorithm [1879] for finding a K-coloring of a graph
• Assume K=3
• Step 1 (simplify): find a node with at most K-1 edges and remove from the graph (with its edges).
(Remember this node on a stack for later stages.)
Coloring a graph

- Once a coloring is found for the simpler graph, we can always color the node we saved on the stack

- **Step 2 (color):** when the simplified subgraph has been colored, add back the node on the top of the stack and assign it a color not taken by one of the adjacent nodes
Coloring with $K=2$

stack:
Coloring

color  register

R1
R2

stack:

c

a
b
d
e
c

Coloring

```
<table>
<thead>
<tr>
<th>color</th>
<th>register</th>
</tr>
</thead>
<tbody>
<tr>
<td>R1</td>
<td></td>
</tr>
<tr>
<td>R2</td>
<td></td>
</tr>
</tbody>
</table>
```

```
stack:

a
e
c
```
Coloring

color     register

R1
R2

stack:

b
da
e

c

a

b
d
e

c

a

Coloring

color   register

R1
R2

stack:
  d
  b
  a
  e
  c
Coloring

color | register
---- | ----
[Blue] | R1
[Red]  | R2

stack:

a
e
c
Coloring

color register

- R1
- R2

The stack contains:
- e
- c
Failure

- If the graph cannot be colored, it will eventually be simplified to a graph in which every node has at least $K$ neighbors.
- Sometimes, the graph is still $K$-colorable!
- Finding a $K$-coloring in all situations is an NP-complete problem.
  - We will have to approximate to make register allocators fast enough.
Coloring with $K=2$

stack:

```
d
```

```
<table>
<thead>
<tr>
<th>color</th>
<th>register</th>
</tr>
</thead>
<tbody>
<tr>
<td>R1</td>
<td></td>
</tr>
<tr>
<td>R2</td>
<td></td>
</tr>
</tbody>
</table>
```

```
      a
     /\  \
    /   \
   b   c
```

```
e
```

Coloring

all nodes have 2 neighbours!

stack: d

color register

R1

R2
Coloring

stack:

b
d
Coloring

- **color**: register
  - R1
  - R2

- **stack**: e a b d
Coloring

Coloring

<table>
<thead>
<tr>
<th>color</th>
<th>register</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>R1</td>
</tr>
<tr>
<td></td>
<td>R2</td>
</tr>
</tbody>
</table>

Stack:

- a
- b
- d

Graph:

- a -> b
- b -> d
- b -> e
- c
- a
Coloring

color register

R1  R2

color     register

stack:

b
d
Coloring

Stack:

R1

R2

Color

Register

d
Coloring

We got lucky!
Try to Color this with 4 colors? 3 colors?
One Possible 4 coloring
The code would look like this...

```
r_1 := 2 * r_2
r_2 := r_3 + r_4
r_3 := -r_2
r_2 := r_2 - 1
r_3 := r_1 + r_4
```
Coloring with $K=2$

Some graphs can’t be colored in $K$ colors:

```
stack:
  c
  b
  e
  a
  d
```
Some graphs can’t be colored in K colors:

stack:

b
ea
d
Some graphs can’t be colored in K colors:
Some graphs can’t be colored in $K$ colors:

- No colors left for $e$!
• **Step 3 (spilling):** once all nodes have $K$ or more neighbors, pick a node for spilling
  – Store on the stack
• There are many heuristics that can be used to pick a node
  – E.g., not in an inner loop
Spilling: Inserting Code

- Since optimistic coloring failed we must spill temporary $f$
- We must allocate a memory location as the home of $f$
  - Typically this is in the current stack frame
  - Call this address $fa$
- Before each operation that uses $f$, insert
  $f := \text{load } fa$
- After each operation that defines $f$, insert
  $\text{store } f, fa$
Example

\[
\begin{align*}
  a & := b + c \\
  d & := -a \\
  f & := \text{load } fa \\
  e & := d + f \\
  f & := 2 \times e \\
  \text{store } f, fa \\
  b & := d + e \\
  e & := e - 1 \\
  f & := \text{load } fa \\
  b & := f + c
\end{align*}
\]
Recomputing Variable Liveness

\[
\begin{align*}
    a & := b + c \\
    d & := -a \\
    f & := \text{load } fa \\
    e & := d + f \\
    f & := 2 \times e \\
    \text{store } f, \text{ fa} \\
    b & := d + e \\
    e & := e - 1
\end{align*}
\]
Recompute the RIG after spilling

This is 3-colorable!
Overall Algorithm

1. Simplify
2. Mark possible spills
3. Color & detect actual spills
4. Liveness
5. Rewrite code to implement actual spills
• Register allocation has three major parts

  – Liveness analysis
  – Graph coloring
  – Program transformation (spilling)

• For more information, chapter 11.1-11.3 in Appel