Register Allocation
Announcements

• Programming Project 4 due Saturday, August 18 at 11:30AM
  • OH all this week.
  • Ask questions via email!
  • Ask questions via Piazza!
  • No late submissions.
Please evaluate this course on Axess.

Your feedback really makes a difference.
## Where We Are

<table>
<thead>
<tr>
<th>Source Code</th>
<th>Lexical Analysis</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Syntax Analysis</td>
</tr>
<tr>
<td></td>
<td>Semantic Analysis</td>
</tr>
<tr>
<td></td>
<td>IR Generation</td>
</tr>
<tr>
<td></td>
<td><strong>IR Optimization</strong></td>
</tr>
<tr>
<td></td>
<td>Code Generation</td>
</tr>
<tr>
<td></td>
<td>Optimization</td>
</tr>
</tbody>
</table>

**Machine Code**
Where We Are

Source Code

Lexical Analysis
Syntax Analysis
Semantic Analysis
IR Generation
IR Optimization
Code Generation

Machine Code

Achievement unlocked
Fan-TAC-stic!
Code Generation at a Glance

• At this point, we have optimized IR code that needs to be converted into the target language (e.g. assembly, machine code).

• Goal of this stage:
  • Choose the appropriate machine instructions for each IR instruction.
  • Divvy up finite machine resources (registers, caches, etc.)
  • Implement low-level details of the runtime environment.

• Machine-specific optimizations are often done here, though some are treated as part of a final optimization phase.
Overview

- **Register Allocation** (Today)
  - How to assign variables to finitely many registers?
  - What to do when it can't be done?
  - How to do so efficiently?

- **Garbage Collection** (Monday)
  - How to detect reclaimable memory?
  - How to reclaim memory efficiently?
Memory Tradeoffs

- There is an enormous tradeoff between speed and size in memory.
- SRAM is fast but very expensive:
  - Can keep up with processor speeds in the GHz.
  - As of 2007, cost is $10/MB
  - Good luck buying 1TB of the stuff!
- Hard disks are cheap but very slow:
  - As of 2012, you can buy a 2TB hard drive for about $100
  - As of 2012, good disk seek times are measured in ms
    (about two to four million times slower than a processor cycle!)
The Memory Hierarchy

• **Idea**: Try to get the best of all worlds by using multiple types of memory.
The Memory Hierarchy

- **Idea**: Try to get the best of all worlds by using multiple types of memory.
The Memory Hierarchy

- **Idea**: Try to get the best of all worlds by using multiple types of memory.
The Memory Hierarchy

- **Idea**: Try to get the best of all worlds by using multiple types of memory.
The Challenges of Code Generation

- Almost all programming languages expose a coarse view of the memory hierarchy:
  - All variables live in “memory.”
  - Disk and network explicitly handled separately.
- (Interesting exception: Stanford's Sequoia programming language)
- Challenges in code generation:
  - Position objects in a way that takes maximum advantage of the memory hierarchy.
  - Do so without hints from the programmer.
Registers

• Most machines have a set of registers, dedicated memory locations that
  • can be accessed quickly,
  • can have computations performed on them, and
  • exist in small quantity.

• Using registers intelligently is a critical step in any compiler.
  • A good register allocator can generate code orders of magnitude better than a bad register allocator.
Register Allocation

• In TAC, there are an unlimited number of variables.
• On a physical machine there are a small number of registers:
  • x86 has four general-purpose registers and a number of specialized registers.
  • MIPS has twenty-four general-purpose registers and eight special-purpose registers.
• **Register allocation** is the process of assigning variables to registers and managing data transfer in and out of registers.
Challenges in Register Allocation

- **Registers are scarce.**
  - Often substantially more IR variables than registers.
  - Need to find a way to reuse registers whenever possible.

- **Registers are complicated.**
  - x86: Each register made of several smaller registers; can't use a register and its constituent registers at the same time.
  - x86: Certain instructions must store their results in specific registers; can't store values there if you want to use those instructions.
  - MIPS: Some registers reserved for the assembler or operating system.
  - Most architectures: Some registers must be preserved across function calls.
Goals for Today

• Introduce register allocation for a MIPS-style machine:
  • Some number of indivisible, general-purpose registers.
• Explore three algorithms for register allocation:
  • Naïve ("no") register allocation.
  • Linear scan register allocation.
  • Graph-coloring register allocation.
An Initial Register Allocator

• **Idea**: Store every value in main memory, loading values only when they're needed.

• To generate a code that performs a computation:
  • Generate **load** instructions to pull the values from main memory into registers.
  • Generate code to perform the computation on the registers.
  • Generate **store** instructions to store the result back into main memory.
Our Register Allocator In Action
Our Register Allocator In Action

\[ a = b + c; \]
\[ d = a; \]
\[ c = a + d; \]
Our Register Allocator In Action

\[
\begin{align*}
    a & = b + c; \\
    d & = a; \\
    c & = a + d;
\end{align*}
\]

<table>
<thead>
<tr>
<th>Param N</th>
<th>fp + 4N</th>
</tr>
</thead>
<tbody>
<tr>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>Param 1</td>
<td>fp + 4</td>
</tr>
<tr>
<td>Stored fp</td>
<td>fp + 0</td>
</tr>
<tr>
<td>Stored ra</td>
<td>fp - 4</td>
</tr>
<tr>
<td>a</td>
<td>fp - 8</td>
</tr>
<tr>
<td>b</td>
<td>fp - 12</td>
</tr>
<tr>
<td>c</td>
<td>fp - 16</td>
</tr>
<tr>
<td>d</td>
<td>fp - 20</td>
</tr>
</tbody>
</table>
Our Register Allocator In Action

\[ a = b + c; \]
\[ d = a; \]
\[ c = a + d; \]
Our Register Allocator In Action

\[
a = b + c; \quad \text{lw} \quad \$t0, \quad -12(fp)
d = a;
c = a + d;
\]

<table>
<thead>
<tr>
<th>Param N</th>
<th>fp + 4N</th>
</tr>
</thead>
<tbody>
<tr>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>Param 1</td>
<td>fp + 4</td>
</tr>
<tr>
<td>Stored fp</td>
<td>fp + 0</td>
</tr>
<tr>
<td>Stored ra</td>
<td>fp - 4</td>
</tr>
<tr>
<td>a</td>
<td>fp - 8</td>
</tr>
<tr>
<td>b</td>
<td>fp - 12</td>
</tr>
<tr>
<td>c</td>
<td>fp - 16</td>
</tr>
<tr>
<td>d</td>
<td>fp - 20</td>
</tr>
</tbody>
</table>
Our Register Allocator In Action

\[
a = b + c; \\
d = a; \\
c = a + d;
\]

\[
lw \ $t0, \ -12(fp) \\
lw \ $t1, \ -16(fp)
\]
Our Register Allocator In Action

\[
\begin{align*}
a &= b + c; \\
d &= a; \\
c &= a + d;
\end{align*}
\]

\[
\text{lw } \$t0, \ -12(fp) \\
\text{lw } \$t1, \ -16(fp) \\
\text{add } \$t2, \ \$t0, \ \$t1
\]

<table>
<thead>
<tr>
<th>Param N</th>
<th>fp + 4N</th>
</tr>
</thead>
<tbody>
<tr>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>Param 1</td>
<td>fp + 4</td>
</tr>
<tr>
<td>Stored fp</td>
<td>fp + 0</td>
</tr>
<tr>
<td>Stored ra</td>
<td>fp - 4</td>
</tr>
<tr>
<td>a</td>
<td>fp - 8</td>
</tr>
<tr>
<td>b</td>
<td>fp - 12</td>
</tr>
<tr>
<td>c</td>
<td>fp - 16</td>
</tr>
<tr>
<td>d</td>
<td>fp - 20</td>
</tr>
</tbody>
</table>
Our Register Allocator In Action

\[ a = b + c; \]
\[ d = a; \]
\[ c = a + d; \]

**Param N**

- fp + 4N
- fp + 4
- fp - 4
- fp - 8
- fp - 12
- fp - 16
- fp - 20

**Stored fp**

- fp + 0

**Stored ra**

- fp - 4

```
lw $t0, -12(fp)
lw $t1, -16(fp)
add $t2, $t0, $t1
sw $t2, -8(fp)
```
Our Register Allocator In Action

\[ a = b + c; \]
\[ d = a; \]
\[ c = a + d; \]

```
lw  $t0,  -12(fp)
lw  $t1,  -16(fp)
add $t2,  $t0,  $t1
sw   $t2,  -8(fp)
```
Our Register Allocator In Action

\[
\begin{align*}
    a &= b + c; \\
    d &= a; \\
    c &= a + d; \\
\end{align*}
\]

<table>
<thead>
<tr>
<th>Param N</th>
<th>( fp + 4N )</th>
</tr>
</thead>
<tbody>
<tr>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Param 1</th>
<th>( fp + 4 )</th>
</tr>
</thead>
</table>

<table>
<thead>
<tr>
<th>Stored fp</th>
<th>( fp + 0 )</th>
</tr>
</thead>
</table>

<table>
<thead>
<tr>
<th>Stored ra</th>
<th>( fp - 4 )</th>
</tr>
</thead>
</table>

<table>
<thead>
<tr>
<th>a</th>
<th>( fp - 8 )</th>
</tr>
</thead>
</table>

<table>
<thead>
<tr>
<th>b</th>
<th>( fp - 12 )</th>
</tr>
</thead>
</table>

<table>
<thead>
<tr>
<th>c</th>
<th>( fp - 16 )</th>
</tr>
</thead>
</table>

<table>
<thead>
<tr>
<th>d</th>
<th>( fp - 20 )</th>
</tr>
</thead>
</table>

\[
\begin{align*}
    lw \ & \ t0, -12(fp) \\
    lw \ & \ t1, -16(fp) \\
    add \ & \ t2, t0, t1 \\
    sw \ & \ t2, -8(fp) \\
    lw \ & \ t0, -8(fp)
\end{align*}
\]
Our Register Allocator In Action

\[
a = b + c; \\
d = a; \\
c = a + d;
\]

<table>
<thead>
<tr>
<th>Param N</th>
<th>Stored fp</th>
<th>Stored ra</th>
</tr>
</thead>
<tbody>
<tr>
<td>[fp + 4N]</td>
<td>[fp + 0]</td>
<td>[fp - 4]</td>
</tr>
<tr>
<td>[fp + 4]</td>
<td>[fp - 8]</td>
<td>[fp - 12]</td>
</tr>
<tr>
<td>[fp - 4]</td>
<td>[fp - 16]</td>
<td>[fp - 20]</td>
</tr>
</tbody>
</table>

```
lw  $t0, -12(fp)
lw  $t1, -16(fp)
add $t2, $t0, $t1
sw  $t2, -8(fp)
lw  $t0, -8(fp)
sw  $t0, -20(fp)
```
Our Register Allocator In Action

```plaintext
a = b + c;
d = a;
c = a + d;

lw  $t0,  -12(fp)
lw  $t1,  -16(fp)
add $t2, $t0, $t1
sw  $t2,   -8(fp)
lw  $t0,  -8(fp)
sw  $t0,  -20(fp)
```

<table>
<thead>
<tr>
<th></th>
<th>Param N</th>
<th>Stored fp</th>
<th>Stored ra</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>fp + 4N</td>
<td>fp + 0</td>
<td>fp - 4</td>
</tr>
<tr>
<td>Param 1</td>
<td>...</td>
<td>fp + 4</td>
<td>fp - 8</td>
</tr>
<tr>
<td></td>
<td>...</td>
<td>fp - 4</td>
<td>fp - 12</td>
</tr>
<tr>
<td></td>
<td></td>
<td>fp - 4</td>
<td>fp - 16</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>fp - 20</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Our Register Allocator In Action

\[
\begin{align*}
a &= b + c; \\
d &= a; \\
c &= a + d;
\end{align*}
\]

\[
\begin{align*}
lw & \quad \$t0, \quad -12(fp) \\
lw & \quad \$t1, \quad -16(fp) \\
add & \quad \$t2, \quad \$t0, \quad \$t1 \\
sw & \quad \$t2, \quad \ -8(fp) \\
lw & \quad \$t0, \quad -8(fp) \\
sw & \quad \$t0, \quad -20(fp) \\
lw & \quad \$t0, \quad -8(fp)
\end{align*}
\]
Our Register Allocator In Action

\[ a = b + c; \]
\[ d = a; \]
\[ c = a + d; \]

lw \ $t0, \ -12(fp) \\
lw \ $t1, \ -16(fp) \\
add \ $t2, \ $t0, \ $t1 \\
sw \ $t2, \ -8(fp) \\
lw \ $t0, \ -8(fp) \\
sw \ $t0, \ -20(fp) \\
lw \ $t0, \ -8(fp) \\
lw \ $t1, \ -20(fp)
Our Register Allocator In Action

\[
\begin{align*}
    a &= b + c; \\
    d &= a; \\
    c &= a + d;
\end{align*}
\]

\[
\begin{align*}
    \text{l}w & \; \$t0, \; -12(fp) \\
    \text{l}w & \; \$t1, \; -16(fp) \\
    \text{a}d & \; \$t2, \; \$t0, \; \$t1 \\
    \text{s}w & \; \$t2, \; -8(fp) \\
\end{align*}
\]

<table>
<thead>
<tr>
<th>Param N</th>
<th>\text{fp} + 4N</th>
</tr>
</thead>
<tbody>
<tr>
<td>\ldots</td>
<td>\ldots</td>
</tr>
<tr>
<td>Param 1</td>
<td>\text{fp} + 4</td>
</tr>
<tr>
<td>Stored fp</td>
<td>\text{fp} + 0</td>
</tr>
<tr>
<td>Stored ra</td>
<td>\text{fp} - 4</td>
</tr>
<tr>
<td>a</td>
<td>\text{fp} - 8</td>
</tr>
<tr>
<td>b</td>
<td>\text{fp} - 12</td>
</tr>
<tr>
<td>c</td>
<td>\text{fp} - 16</td>
</tr>
<tr>
<td>d</td>
<td>\text{fp} - 20</td>
</tr>
</tbody>
</table>
Our Register Allocator In Action

\[ a = b + c; \]
\[ d = a; \]
\[ c = a + d; \]

```
lw  $t0, -12(fp)
lw  $t1, -16(fp)
add $t2, $t0, $t1
sw  $t2, -8(fp)
```

<table>
<thead>
<tr>
<th>Param N</th>
<th>Stored fp</th>
<th>Stored ra</th>
</tr>
</thead>
<tbody>
<tr>
<td>fp + 4N</td>
<td>fp + 0</td>
<td>fp - 4</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>fp - 8</td>
</tr>
<tr>
<td>fp + 4</td>
<td>fp - 4</td>
<td>...</td>
</tr>
<tr>
<td>fp - 8</td>
<td>fp - 4</td>
<td>fp - 12</td>
</tr>
<tr>
<td>fp - 12</td>
<td>fp - 8</td>
<td>fp - 16</td>
</tr>
<tr>
<td>fp - 16</td>
<td>fp - 12</td>
<td>fp - 20</td>
</tr>
<tr>
<td>fp - 20</td>
<td>fp - 16</td>
<td>fp - 20</td>
</tr>
</tbody>
</table>

```Param 1```
Analysis of our Allocator

- Disadvantage: **Gross inefficiency**.
  - Issues unnecessary loads and stores by the dozen.
  - Wastes space on values that could be stored purely in registers.
  - Easily an order of magnitude or two slower than necessary.
  - Unacceptable in any production compiler.

- Advantage: **Simplicity**.
  - Can translate each piece of IR directly to assembly as we go.
  - Never need to worry about running out of registers.
  - Never need to worry about function calls or special-purpose registers.
  - Good if you just needed to get a prototype compiler up and running.
Building a Better Allocator

- **Goal**: Try to hold as many variables in registers as possible.
  - Reduces memory reads/writes.
  - Reduces total memory usage.
- We will need to address these questions:
  - Which registers do we put variables in?
  - What do we do when we run out of registers?
Register Consistency

\[
a = b + c
\]

\[
d = a
\]

\[
d = b
\]

\[
e = d
\]
Register Consistency

• At each program point, each variable must be in the same location.
  • Does **not** mean that each variable is always stored in the same location!

• At each program point, each register holds at most one live variable.
  • Can assign several variables the same register if no two of them ever will be read together.
Live Ranges and Live Intervals

- Recall: A variable is **live** at a particular program point if its value may be read later before it is written.
  - Can find this using global liveness analysis.
- The **live range** for a variable is the set of program points at which that variable is live.
- The **live interval** for a variable is the smallest subrange of the IR code containing all a variable's live ranges.
  - A property of the IR code, **not** the CFG.
  - Less precise than live ranges, but simpler to work with.
Live Ranges and Live Intervals
Live Ranges and Live Intervals

\[
\begin{align*}
e &= d + a \\
f &= b + c \\
f &= f + b \\
\text{IfZ } e & \text{ Goto } _L0 \\
d &= e + f \\
\text{Goto } _L1; \\
_L0: \\
\hspace{1em} d &= e - f \\
_L1: \\
\hspace{1em} g &= d
\end{align*}
\]
Live Ranges and Live Intervals

\[
\begin{align*}
e & = d + a \\
f & = b + c \\
f & = f + b \\
\text{IfZ } e & \text{ Goto } _{L0} \\
d & = e + f \\
\text{Goto } _{L1} ; \\
_{L0} : \\
& \quad d = e - f \\
_{L1} : \\
& \quad g = d \\
\end{align*}
\]
Live Ranges and Live Intervals

\[ e = d + a \]
\[ f = b + c \]
\[ f = f + b \]
\[ \text{IfZ } e \text{ Goto } _{L0} \]
\[ d = e + f \]
\[ \text{Goto } _{L1}; \]
\[ _{L0}: \]
\[ d = e - f \]
\[ _{L1}: \]
\[ g = d \]
\[ g = d \]
\{ g \}
Live Ranges and Live Intervals

\[
e = d + a
\]

\[
f = b + c
\]

\[
f = f + b
\]

\[
\text{IfZ e Goto _L0}
\]

\[
d = e + f
\]

\[
\text{Goto _L1;}
\]

_L0:

\[
d = e - f
\]

_L1:

\[
g = d
\]

\[
e = d + a
\]

\[
f = b + c
\]

\[
f = f + b
\]

_L0:

\[
d = e + f
\]

\[
d = e - f
\]

{ d }

{ g }
e = d + a
f = b + c
f = f + b
IfZ e Goto _L0
d = e + f
Goto _L1;
)L0:
d = e - f
)L1:
g = d

\[ e = d + a \]
\[ f = b + c \]
\[ f = f + b \]
\[ \text{IfZ } e \text{ Goto } _L0 \]
\[ d = e + f \]
\[ \text{Goto } _L1; \]
\[ _L0: \]
\[ d = e - f \]
\[ _L1: \]
\[ g = d \]
\[ e = d + a \]
\[ f = b + c \]
\[ f = f + b \]

IfZ e Goto _L0
\[ d = e + f \]
Goto _L1;

_L0:
\[ d = e - f \]

_L1:
\[ g = d \]
\[ e = d + a \]
\[ f = b + c \]
\[ f = f + b \]
\[ \text{IfZ } e \text{ Goto } _L0 \]
\[ d = e + f \]
\[ \text{Goto } _L1; \]
\[ _L0: \]
\[ d = e - f \]
\[ _L1: \]
\[ g = d \]
\[
e = d + a \\
f = b + c \\
f = f + b \\
\text{IfZ } e \text{ Goto } _L0 \\
d = e + f \\
\text{Goto } _L1; \\
_L0: \\
\quad d = e - f \\
_L1: \\
\quad g = d
\]
e = d + a
f = b + c
f = f + b
IfZ e Goto _L0
d = e + f
Goto _L1;
_L0:
d = e - f
_L1:
g = d

e = d + a
f = b + c
f = f + b
{ e, f }
Live Ranges and Live Intervals

e = d + a
f = b + c
f = f + b
IfZ e Goto _L0
d = e + f
Goto _L1;
_L0:
d = e - f
_L1:
g = d
Live Ranges and Live Intervals

\[
e = d + a
\]
\[
f = b + c
\]
\[
f = f + b
\]
\[
\text{IfZ } e \text{ Goto } _L0
\]
\[
d = e + f
\]
\[
\text{Goto } _L1;
\]

\_L0:
\[
d = e - f
\]

\_L1:
\[
g = d
\]

\[
e = d + a
\]
\[
f = b + c
\]
\[
\{ b, e, f \}
\]
\[
d = e + f
\]
\[
\{ b, e, f \}
\]
\[
f = f + b
\]
\[
\{ e, f \}
\]
\[
d = e - f
\]
\[
\{ e, f \}
\]
\[
d = e + f
\]
\[
\{ d \}
\]
\[
d = e - f
\]
\[
\{ e, f \}
\]
\[
d = e + f
\]
\[
\{ d \}
\]
\[
g = d
\]
\[
\{ g \}
\]
\begin{align*}
e &= d + a \\
f &= b + c \\
f &= f + b \\
\text{IfZ } e \text{ Goto } _L0 \\
d &= e + f \\
\text{Goto } _L1;
\end{align*}

\_L0:
\begin{align*}
d &= e - f
\end{align*}

\_L1:
\begin{align*}
g &= d
\end{align*}
Live Ranges and Live Intervals

\[
\begin{align*}
e & = d + a \\
f & = b + c \\
f & = f + b \\
\text{IfZ } e \text{ Goto } _L0 \\
d & = e + f \\
\text{Goto } _L1; \\
_L0: & \\
\quad d & = e - f \\
_L1: & \\
\quad g & = d
\end{align*}
\]
Live Ranges and Live Intervals

\[
e = d + a
\]
\[
f = b + c
\]
\[
f = f + b
\]
\[
\text{IfZ } e \text{ Goto } _L0
\]
\[
d = e + f
\]
\[
\text{Goto } _L1;
\]

_ L0:
\[
d = e - f
\]

_ L1:
\[
g = d
\]
Live Ranges and Live Intervals

\[
\begin{align*}
e &= d + a \\
f &= b + c \\
f &= f + b \\
\text{IfZ } e \text{ Goto } _L0 \\
d &= e + f \\
\text{Goto } _L1; \\
_L0: \\
d &= e - f \\
_L1: \\
g &= d
\end{align*}
\]

\[
\begin{align*}
\{ a, b, c, d \} \\
e &= d + a \\
\{ b, c, e \}
\end{align*}
\]

\[
\begin{align*}
\{ b, c, e \} \\
f &= b + c \\
\{ b, e, f \}
\end{align*}
\]

\[
\begin{align*}
\{ b, e, f \} \\
f &= f + b \\
\{ e, f \}
\end{align*}
\]

\[
\begin{align*}
\{ e, f \} \\
d &= e + f \\
\{ d \}
\end{align*}
\]

\[
\begin{align*}
\{ e, f \} \\
d &= e - f \\
\{ d \}
\end{align*}
\]

\[
\begin{align*}
\{ d \} \\
g &= d \\
\{ g \}
\end{align*}
\]
Live Ranges and Live Intervals

```
{ a, b, c, d }
  e = d + a
  { b, c, e }

{ b, c, e }
  f = b + c
  f = f + b

IfZ e Goto _L0
{ b, c, e }
  f = b + c
  { b, e, f }

_d0:
  d = e + f
  Goto _L1;

_L1:
  f = f + b
  { e, f }

{ e, f }
  d = e + f
  { d }
{ e, f }
  d = e - f
  { d }

{ d }
g = d
  { g }
```

Live Ranges and Live Intervals

\[
\begin{align*}
\text{e} &= \text{d} + \text{a} \\
\text{f} &= \text{b} + \text{c} \\
\text{f} &= \text{f} + \text{b} \\
\text{IfZ e Goto } &\_L0 \\
\text{d} &= \text{e} + \text{f} \\
\text{Goto } &\_L1; \\
\_L0: \\
\text{d} &= \text{e} - \text{f} \\
\_L1: \\
\text{g} &= \text{d}
\end{align*}
\]
Live Ranges and Live Intervals

\begin{align*}
e &= d + a \\
f &= b + c \\
f &= f + b \\
\text{IfZ } e \text{ Goto } _L0 \\
d &= e + f \\
\text{Goto } _L1; \\
\_L0: \\
d &= e - f \\
\_L1: \\
g &= d
\end{align*}
Live Ranges and Live Intervals

```
e = d + a
f = b + c
f = f + b
IfZ e Goto _L0
d = e + f
Goto _L1;
_L0:
d = e - f
_L1:
g = d
```

```
{ a, b, c, d }
e = d + a
{ b, c, e }

{ b, c, e }
f = b + c
{ b, e, f }

{ b, e, f }
f = f + b
{ e, f }

{ e, f }
d = e + f
{ d }

{ e, f }
d = e - f
{ d }

{ d }
g = d
{ g }
```
### Live Ranges and Live Intervals

<table>
<thead>
<tr>
<th>a</th>
<th>b</th>
<th>c</th>
<th>d</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td>{ a, b, c, d }</td>
</tr>
</tbody>
</table>

\[ e = d + a \]

\[ f = b + c \]

\[ f = f + b \]

IfZ e Goto _L0

\[ d = e + f \]

Goto _L1;

_L0:

\[ d = e - f \]

_L1:

\[ g = d \]

\[ e = d + a \] -> \{ b, c, e \}

\[ f = b + c \] -> \{ b, c, e \}

\[ f = f + b \] -> \{ b, c, e \}

IfZ e Goto _L0

\[ d = e + f \] -> \{ b, e, f \}

\[ f = f + b \] -> \{ e, f \}

\[ e = d + a \] -> \{ e, f \}

\[ f = b + c \] -> \{ e, f \}

\[ f = f + b \] -> \{ e, f \}

\[ d = e + f \] -> \{ e, f \}

\[ d = e - f \] -> \{ e, f \}

\[ g = d \] -> \{ d \}

\[ d = e + f \] -> \{ d \}

\[ d = e - f \] -> \{ d \}

\[ g = d \] -> \{ g \}
Live Ranges and Live Intervals

\begin{align*}
e &= d + a \\
f &= b + c \\
f &= f + b \\
&\text{IfZ } e \text{ Goto } _L0 \\
d &= e + f \\
&\text{Goto } _L1; \\
&_L0: \\
&d = e - f \\
&_L1: \\
g &= d
\end{align*}
Live Ranges and Live Intervals

\[ e = d + a \]
\[ f = b + c \]
\[ f = f + b \]
\[ \text{IfZ } e \text{ Goto } _L0 \]
\[ d = e + f \]
\[ \text{Goto } _L1; \]

\_L0:
\[ d = e - f \]

\_L1:
\[ g = d \]

\{ a, b, c, d \}
\[ e = d + a \]
\[ \{ b, c, e \} \]

\{ b, c, e \}
\[ f = b + c \]
\[ \{ b, e, f \} \]

\{ b, c, e \}
\[ f = f + b \]
\[ \{ e, f \} \]

\{ e, f \}
\[ d = e + f \]
\[ \{ d \} \]

\{ e, f \}
\[ d = e - f \]
\[ \{ d \} \]

\{ d \}
\[ g = d \]
\[ \{ g \} \]
Live Ranges and Live Intervals

\[
\begin{align*}
\text{e} &= \text{d} + \text{a} \\
\text{f} &= \text{b} + \text{c} \\
\text{f} &= \text{f} + \text{b} \\
\text{IfZ e Goto } \_\text{L0} \\
\text{d} &= \text{e} + \text{f} \\
\text{Goto } \_\text{L1;} \\
\_\text{L0:} \\
\text{d} &= \text{e} - \text{f} \\
\_\text{L1:} \\
\text{g} &= \text{d}
\end{align*}
\]
Register Allocation with Live Intervals

- Given the live intervals for all the variables in the program, we can allocate registers using a simple greedy algorithm.

- Idea: Track which registers are free at each point.

- When a live interval begins, give that variable a free register.

- When a live interval ends, the register is once again free.

- We can't always fit everything into a register; we'll see what do to in a minute.
Register Allocation with Live Intervals
Register Allocation with Live Intervals

Free Registers

R₀ R₁ R₂ R₃
Register Allocation with Live Intervals

Free Registers

R₀  R₁  R₂  R₃
Register Allocation with Live Intervals

```
| a | b | c | d | e | f | g |
```

Free Registers

```
R_0  R_1  R_2  R_3
```
Register Allocation with Live Intervals

Free Registers

R₀ | R₁ | R₂ | R₃
Register Allocation with Live Intervals
Register Allocation with Live Intervals

Free Registers

- $R_0$
- $R_1$
- $R_2$
- $R_2$
Register Allocation with Live Intervals

Free Registers

\[
\begin{array}{cccc}
R_0 & R_1 & R_2 & R_2 \\
\end{array}
\]
Register Allocation with Live Intervals

Free Registers

<table>
<thead>
<tr>
<th>R_0</th>
<th>R_1</th>
<th>R_2</th>
<th>R_2</th>
</tr>
</thead>
</table>

abc
def
g
Register Allocation with Live Intervals

Free Registers

<table>
<thead>
<tr>
<th>R₀</th>
<th>R₁</th>
<th>R₂</th>
<th>R₂</th>
</tr>
</thead>
</table>

Register Allocation with Live Intervals

Free Registers

R_0  R_1  R_2  R_3
Register Allocation with Live Intervals

Free Registers:

- \( R_0 \)
- \( R_1 \)
- \( R_2 \)
- \( R_3 \)
Register Allocation with Live Intervals

Free Registers

R₀  R₁  R₂  R₃

a  b  c  d  e  f  g
Another Example
Another Example
Another Example

Free Registers

R₀  R₁  R₂
Another Example

Free Registers

R₀  R₁  R₂
Another Example

Free Registers

R₀  R₁  R₂
Another Example

<table>
<thead>
<tr>
<th>a</th>
<th>b</th>
<th>c</th>
<th>d</th>
<th>e</th>
<th>f</th>
<th>g</th>
</tr>
</thead>
</table>

Free Registers

| R₀ | R₁ | R₂ |
Another Example

Free Registers

<table>
<thead>
<tr>
<th>(R_0)</th>
<th>(R_1)</th>
<th>(R_2)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

a b c d e f g
Another Example

Free Registers

What do we do now?
Register Spilling

• If a register cannot be found for a variable \( v \), we may need to \textbf{spill} a variable.

• When a variable is spilled, it is stored in memory rather than a register.

• When we need a register for the spilled variable:
  • Evict some existing register to memory.
  • Load the variable into the register.
  • When done, write the register back to memory and reload the register with its original value.

• Spilling is slow, but sometimes necessary.
Another Example

What do we do now?
Another Example

Free Registers

\[ R_0 \quad R_1 \quad R_2 \]
Another Example

Free Registers

<table>
<thead>
<tr>
<th>R₀</th>
<th>R₁</th>
<th>R₂</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

abcdefg
Another Example

Free Registers

<table>
<thead>
<tr>
<th></th>
<th>R₀</th>
<th>R₁</th>
<th>R₂</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>b</td>
<td>c</td>
<td>d</td>
</tr>
<tr>
<td>e</td>
<td>f</td>
<td>g</td>
<td></td>
</tr>
</tbody>
</table>
Another Example

Free Registers

<table>
<thead>
<tr>
<th>R₀</th>
<th>R₁</th>
<th>R₂</th>
</tr>
</thead>
</table>

a b c d e f g
Another Example

Free Registers

R₀  R₁  R₂
Another Example

<table>
<thead>
<tr>
<th>Free Registers</th>
</tr>
</thead>
<tbody>
<tr>
<td>R₀</td>
</tr>
</tbody>
</table>

Diagram with registers a, b, c, d, e, f, g.
Another Example

Free Registers

<table>
<thead>
<tr>
<th></th>
<th>R₀</th>
<th>R₁</th>
<th>R₂</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Another Example

Free Registers

<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>$R_0$</td>
<td>$R_1$</td>
<td>$R_2$</td>
</tr>
</tbody>
</table>

Diagram showing the allocation of registers with letters a to g.
Another Example

<table>
<thead>
<tr>
<th>a</th>
<th>b</th>
<th>c</th>
<th>d</th>
<th>e</th>
<th>f</th>
<th>g</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Free Registers

<table>
<thead>
<tr>
<th>R₀</th>
<th>R₁</th>
<th>R₂</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Linear Scan Register Allocation

- This algorithm is called **linear scan register allocation** and is a comparatively new algorithm.

- Advantages:
  - Very efficient (after computing live intervals, runs in linear time)
  - Produces good code in many instances.
  - Allocation step works in one pass; can generate code during iteration.
  - Often used in JIT compilers like Java HotSpot.

- Disadvantages:
  - Imprecise due to use of live **intervals** rather than live **ranges**.
  - Other techniques known to be superior in many cases.
Correctness Proof Sketch

• No register holds two live variables at once:
  • Live intervals are conservative approximations of live ranges.
  • No two variables with overlapping live ranges placed in the same register.

• At each program point, every variable is in the same location:
  • All variables assigned a unique location.
Second-Chance Bin Packing

- A more aggressive version of linear-scan.
- Uses live *ranges* instead of live *intervals*.
- If a variable must be spilled, don't spill all uses of it.
  - A later live range might still fit into a register.
- Requires a final data-flow analysis to confirm variables are assigned consistent locations.
- See “Quality and Speed in Linear-scan Register Allocation” by Traub, Holloway, and Smith.
Second-Chance Bin Packing

a b c d e f g

Free Registers

R₀ R₁ R₂
Second-Chance Bin Packing

Free Registers

$R_0$ $R_1$ $R_2$
Second-Chance Bin Packing
Second-Chance Bin Packing

Free Registers

<table>
<thead>
<tr>
<th>R₀</th>
<th>R₁</th>
<th>R₂</th>
</tr>
</thead>
</table>

abcdefg
Second-Chance Bin Packing

Free Registers

<table>
<thead>
<tr>
<th>R₀</th>
<th>R₁</th>
<th>R₂</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Second-Chance Bin Packing

Free Registers

<table>
<thead>
<tr>
<th>R₀</th>
<th>R₁</th>
<th>R₂</th>
</tr>
</thead>
</table>

a b c d e f g
Second-Chance Bin Packing

Free Registers

R₀ | R₁ | R₂
Second-Chance Bin Packing

Free Registers

<table>
<thead>
<tr>
<th>R₀</th>
<th>R₁</th>
<th>R₂</th>
</tr>
</thead>
</table>

```
abc def g
```

Diagram showing allocation of registers with free registers table.
Second-Chance Bin Packing

Free Registers

<table>
<thead>
<tr>
<th></th>
<th>R₀</th>
<th>R₁</th>
<th>R₂</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>b</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>c</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>d</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>e</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>f</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>g</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Second-Chance Bin Packing

Free Registers

<table>
<thead>
<tr>
<th></th>
<th>R₀</th>
<th>R₁</th>
<th>R₂</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>²</td>
<td></td>
<td></td>
</tr>
<tr>
<td>b</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>c</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>d</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>e</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>f</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>g</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Second-Chance Bin Packing

Free Registers

\[
\begin{array}{ccc}
R_0 & R_1 & R_2 \\
\end{array}
\]
Second-Chance Bin Packing

Free Registers

<table>
<thead>
<tr>
<th></th>
<th>R₀</th>
<th>R₁</th>
<th>R₂</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>b</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>c</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>d</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>e</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>f</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>g</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Second-Chance Bin Packing

Free Registers

<table>
<thead>
<tr>
<th></th>
<th>R₀</th>
<th>R₁</th>
<th>R₂</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Second-Chance Bin Packing

Free Registers

\[
\begin{array}{c|c|c|c}
R_0 & R_1 & R_2 \\
\end{array}
\]
Second-Chance Bin Packing

Free Registers

<table>
<thead>
<tr>
<th>R₀</th>
<th>R₁</th>
<th>R₂</th>
</tr>
</thead>
</table>

a b c d e f g
Second-Chance Bin Packing

Free Registers

<table>
<thead>
<tr>
<th>R₀</th>
<th>R₁</th>
<th>R₂</th>
</tr>
</thead>
</table>

abcdefg
Second-Chance Bin Packing

Free Registers

\[
\begin{array}{c|c|c}
R_0 & R_1 & R_2 \\
\end{array}
\]
Second-Chance Bin Packing

Free Registers

<table>
<thead>
<tr>
<th></th>
<th>R₀</th>
<th>R₁</th>
<th>R₂</th>
</tr>
</thead>
</table>

a b c d e f g
Second-Chance Bin Packing

Free Registers

<table>
<thead>
<tr>
<th>R_0</th>
<th>R_1</th>
<th>R_2</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
An Entirely Different Approach
An Entirely Different Approach

\begin{align*}
\{ a, b, c, d \} \\
e &= d + a \\
\{ b, c, e \} \\
\{ b, c, e \} \\
f &= b + c \\
\{ b, e, f \} \\
\{ b, e, f \} \\
f &= f + b \\
\{ e, f \} \\
\{ e, f \} \\
d &= e + f \\
\{ d \} \\
\{ e, f \} \\
\{ e, f \} \\
d &= e - f \\
\{ d \} \\
\{ d \} \\
g &= d \\
\{ g \} \\
\end{align*}
An Entirely Different Approach

What can we infer from all these variables being live at this point?
An Entirely Different Approach

{ a, b, c, d }
e = d + a
{ b, c, e }

{ b, c, e }
f = b + c
{ b, e, f }

{ b, e, f }
f = f + b
{ e, f }

{ e, f }
d = e + f
{ d }

{ e, f }
d = e - f
{ d }

{ d }
g = d
{ g }
An Entirely Different Approach

{ a, b, c, d }
  e = d + a
  { b, c, e }

{ b, c, e }
  f = b + c
  { b, e, f }

{ b, e, f }
  f = f + b
  { e, f }

{ e, f }
  d = e + f
  { d }

{ e, f }
  d = e - f
  { d }

{ d }
  g = d
  { g }

{ b }

{ f }
An Entirely Different Approach

\[
\begin{align*}
\{ a, b, c, d \} \\
e &= d + a \\
\{ b, c, e \} \\
\{ b, c, e \} \\
f &= b + c \\
\{ b, e, f \} \\
\{ b, e, f \} \\
f &= f + b \\
\{ e, f \} \\
d &= e + f \\
\{ e, f \} \\
\{ d \} \\
\{ e, f \} \\
\{ e, f \} \\
d &= e - f \\
\{ d \} \\
\{ d \} \\
g &= d \\
\{ g \}
\end{align*}
\]
An Entirely Different Approach

\{ a, b, c, d \}
\begin{align*}
e &= d + a \\
\{ b, c, e \}\end{align*}

\{ b, c, e \}
\begin{align*}
f &= b + c \\
\{ b, e, f \}\end{align*}

\{ b, e, f \}
\begin{align*}
f &= f + b \\
\{ e, f \}\end{align*}

\{ e, f \}
\begin{align*}
d &= e + f \\
\{ d \}\end{align*}

\{ e, f \}
\begin{align*}
d &= e - f \\
\{ d \}\end{align*}

\{ d \}
\begin{align*}
g &= d \\
\{ g \}\end{align*}
An Entirely Different Approach

\[
\begin{align*}
\{ a, b, c, d \} \\
& e = d + a \\
\{ b, c, e \} \\
\{ b, c, e \} \\
& f = b + c \\
\{ b, e, f \} \\
\{ b, e, f \} \\
& f = f + b \\
\{ e, f \} \\
\{ e, f \} \\
& d = e + f \\
\{ d \} \\
\{ e, f \} \\
& d = e - f \\
\{ d \} \\
\{ d \} \\
& g = d \\
\{ g \}
\end{align*}
\]
An Entirely Different Approach

\{ a, b, c, d \}
\[
e = d + a
\]
\{ b, c, e \}

\{ b, c, e \}
\[
f = b + c
\]
\{ b, e, f \}

\{ b, e, f \}
\[
f = f + b
\]
\{ e, f \}

\{ e, f \}
\[
d = e + f
\]
\{ d \}

\{ e, f \}
\[
d = e - f
\]
\{ d \}

\{ d \}
\[
g = d
\]
\{ g \}

Registers

\[
R_0 \quad R_1 \quad R_2 \quad R_3
\]
An Entirely Different Approach

\{ a, b, c, d \}
\hspace{1em} e = d + a
\hspace{1em} \{ b, c, e \}

\{ b, c, e \}
\hspace{1em} f = b + c
\hspace{1em} \{ b, e, f \}

\{ b, e, f \}
\hspace{1em} f = f + b
\hspace{1em} \{ e, f \}

\{ e, f \}
\hspace{1em} d = e + f
\hspace{1em} \{ d \}

\{ e, f \}
\hspace{1em} d = e - f
\hspace{1em} \{ d \}

\{ d \}
\hspace{1em} g = d
\hspace{1em} \{ g \}

\{ g \}
\hspace{1em} \{ g \}

Registers

\begin{align*}
R_0 & \quad R_1 \quad R_2 \quad R_3 \\
\end{align*}
The Register Interference Graph

- The **register interference graph** (RIG) of a control-flow graph is an undirected graph where
  - Each node is a variable.
  - There is an edge between two variables that are live at the same program point.
- Perform register allocation by assigning each variable a different register from all of its neighbors.
- There's just one catch...
The One Catch

• This problem is equivalent to graph-coloring, which is NP-hard if there are at least three registers.

• No good polynomial-time algorithms (or even good approximations!) are known for this problem.

• We have to be content with a heuristic that is good enough for RIGs that arise in practice.
The One Catch to The One Catch
If you can figure out a way to assign registers to arbitrary RIGs, you've just proven $P = NP$ and will get a $1,000,000$ check from the Clay Mathematics Institute.
The One Catch to The One Catch

CHALLENGE ACCEPTED

If you can figure out a way to assign registers to arbitrary RIGs, you've just proven $P = NP$ and will get a $1,000,000 check from the Clay Mathematics Institute.
Battling NP-Hardness
Chaitin's Algorithm

- **Intuition:**
  - Suppose we are trying to \( k \)-color a graph and find a node with fewer than \( k \) edges.
  - If we delete this node from the graph and color what remains, we can find a color for this node if we add it back in.
  - Reason: With fewer than \( k \) neighbors, some color must be left over.

- **Algorithm:**
  - Find a node with fewer than \( k \) outgoing edges.
  - Remove it from the graph.
  - Recursively color the rest of the graph.
  - Add the node back in.
  - Assign it a valid color.
Chaitin's Algorithm
Chaitin's Algorithm
Chaitin's Algorithm

Registers

\[ \begin{array}{cccc}
R_0 & R_1 & R_2 & R_3 \\
\end{array} \]
Chaitin's Algorithm

Registers

\[
\begin{array}{c|c|c|c}
R_0 & R_1 & R_2 & R_3 \\
\hline
\text{R0} & \text{R1} & \text{R2} & \text{R3} \\
\end{array}
\]
Chaitin's Algorithm

 Registers

\[ R_0 \quad R_1 \quad R_2 \quad R_3 \]
Chaitin's Algorithm

Registers

\[
\begin{align*}
R_0 & \quad R_1 & \quad R_2 & \quad R_3 \\
\end{align*}
\]
Chaitin's Algorithm

Registers

\[ \begin{array}{cccc}
R_0 & R_1 & R_2 & R_3 \\
\end{array} \]
Chaitin's Algorithm

Registers

\[
\begin{array}{cccc}
R_0 & R_1 & R_2 & R_3 \\
\end{array}
\]
Chaitin's Algorithm

Registers

\[
\begin{array}{cccc}
R_0 & R_1 & R_2 & R_3 \\
\end{array}
\]
Chaitin's Algorithm

Registers

\[R_0\quad R_1\quad R_2\quad R_3\]
Chaitin's Algorithm

Registers

$R_0$ $R_1$ $R_2$ $R_3$
Chaitin's Algorithm

Registers

\[
\begin{array}{cccc}
R_0 & R_1 & R_2 & R_3 \\
\end{array}
\]
Chaitin's Algorithm

 Registers

<table>
<thead>
<tr>
<th>R₀</th>
<th>R₁</th>
<th>R₂</th>
<th>R₃</th>
</tr>
</thead>
</table>

Diagram:

- Nodes: a, b, c, d, e, g, f
- Edges: a→b, a→d, b→c, c→e, d→g, f→g

Nodes:

- a, b, c, d, e, g, f

Registers:

<table>
<thead>
<tr>
<th>R₀</th>
<th>R₁</th>
<th>R₂</th>
<th>R₃</th>
</tr>
</thead>
</table>

(Colors correspond to nodes in the diagram.)
Chaitin's Algorithm

Registers

\[
\begin{array}{cccc}
R_0 & R_1 & R_2 & R_3 \\
\end{array}
\]
Chaitin's Algorithm
Chaitin's Algorithm
Chaitin's Algorithm

Registers

R₀  R₁  R₂  R₃
Chaitin's Algorithm

Registers

\[ \begin{array}{cccc}
R_0 & R_1 & R_2 & R_3 \\
\end{array} \]
Chaitin's Algorithm

Registers

R₀  R₁  R₂  R₃
Chaitin's Algorithm

Registers

\begin{array}{c}
R_0 & R_1 & R_2 & R_3 \\
\end{array}
Chaitin's Algorithm

Registers

\[
\begin{array}{cccc}
R_0 & R_1 & R_2 & R_3 \\
\end{array}
\]
Chaitin's Algorithm

Registers

<table>
<thead>
<tr>
<th>R₀</th>
<th>R₁</th>
<th>R₂</th>
<th>R₃</th>
</tr>
</thead>
</table>
Chaitin's Algorithm

Registers

\[ R_0 \quad R_1 \quad R_2 \quad R_3 \]
Chaitin's Algorithm

Registers

\[
\begin{array}{cccc}
R_0 & R_1 & R_2 & R_3 \\
\end{array}
\]
Chaitin's Algorithm

Registers

| $R_0$ | $R_1$ | $R_2$ | $R_3$ |
Chaitin's Algorithm

Registers

$R_0$ $R_1$ $R_2$ $R_3$
Chaitin's Algorithm

 Registers

\[
\begin{array}{cccc}
R_0 & R_1 & R_2 & R_3 \\
\end{array}
\]
Chaitin's Algorithm

Registers

\begin{tabular}{cccc}
R_0 & R_1 & R_2 & R_3 \\
\end{tabular}
Chaitin's Algorithm

Registers

\begin{tabular}{|c|c|c|c|}
\hline
R_0 & R_1 & R_2 & R_3 \\
\hline
\end{tabular}
One Problem

- What if we can't find a node with fewer than $k$ neighbors?
- Choose and remove an arbitrary node, marking it “troublesome.”
  - Use heuristics to choose which one.
- When adding node back in, it may be possible to find a valid color.
- Otherwise, we have to spill that node.
Chaitin's Algorithm Reloaded

![Diagram of a graph with labeled nodes a, b, c, d, e, f, and g.]

**Registers**

<table>
<thead>
<tr>
<th></th>
<th>R₀</th>
<th>R₁</th>
<th>R₂</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>b</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>c</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>d</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>e</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>f</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>g</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Chaitin's Algorithm Reloaded

Registers

\[ R_0 \quad R_1 \quad R_2 \]
Chaitin's Algorithm Reloaded

Registers

\[ R_0 \quad R_1 \quad R_2 \]
Chaitin's Algorithm Reloaded

Registers

\[ R_0 \quad R_1 \quad R_2 \]
Chaitin's Algorithm Reloaded
Chaitin's Algorithm Reloaded
Chaitin's Algorithm Reloaded

Registers

\[
\begin{array}{ccc}
R_0 & R_1 & R_2 \\
\end{array}
\]
Chaitin's Algorithm Reloaded

Registers

\[
\begin{array}{c|c|c|c}
R_0 & R_1 & R_2 \\
\hline
\end{array}
\]
Chaitin's Algorithm Reloaded

Registers

\[ R_0 \quad R_1 \quad R_2 \]
Chaitin's Algorithm Reloaded

Registers

\begin{align*}
&\text{R}_0 \\
&\text{R}_1 \\
&\text{R}_2
\end{align*}
Chaitin's Algorithm Reloaded
Chaitin's Algorithm Reloaded
Chaitin's Algorithm Reloaded

Registers

\[
R_0 \quad R_1 \quad R_2
\]
Chaitin's Algorithm Reloaded

Registers

- $R_0$
- $R_1$
- $R_2$
Chaitin's Algorithm Reloaded

```
\begin{verbatim}
Registers
R_0  R_1  R_2
\end{verbatim}
```

`d (spilled)`
Chaitin's Algorithm Reloaded

Registers

\[
\begin{array}{c}
R_0 \\
R_1 \\
R_2 \\
\end{array}
\]
Chaitin's Algorithm Reloaded

Registers

\[
\begin{array}{c}
R_0 \\
R_1 \\
R_2
\end{array}
\]
Chaitin's Algorithm Reloaded

Registers

\[
\begin{align*}
R_0 & & R_1 & & R_2 \\
\end{align*}
\]
Chaitin's Algorithm Reloaded
Chaitin's Algorithm Reloaded

Registers

R₀  R₁  R₂
Chaitin's Algorithm Reloaded

Registers

<table>
<thead>
<tr>
<th></th>
<th>R₀</th>
<th>R₁</th>
<th>R₂</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>b</td>
<td>c</td>
<td>d</td>
</tr>
<tr>
<td>e</td>
<td>f</td>
<td>g</td>
<td>(spilled)</td>
</tr>
</tbody>
</table>
A Smarter Algorithm

\[
\begin{align*}
\{ a, b, c, d \} & \quad e = d + a \quad \{ b, c, e \} \\
\{ b, c, e \} & \quad f = b + c \quad \{ b, e, f \} \\
\{ b, e, f \} & \quad f = f + b \quad \{ e, f \} \\
\{ e, f \} & \quad d = e + f \quad \{ d \} \\
\{ e, f \} & \quad d = e - f \quad \{ d \} \\
\{ d \} & \quad g = d \quad \{ g \}
\end{align*}
\]
A Smarter Algorithm

\[
\begin{align*}
\{ a, b, c, d \} \\
e &= d + a \\
\{ b, c, e \} \\
\{ b, c, e \} \\
f &= b + c \\
\{ b, e, f \} \\
f &= f + b \\
\{ e, f \} \\
\{ e, f \} \\
d &= e + f \\
\{ d \} \\
\{ e, f \} \\
d &= e - f \\
\{ d \} \\
\{ d \} \\
\{ d \} \\
\{ g \} \\
g &= d
\end{align*}
\]
A Smarter Algorithm

{ a, b, c, d }
e = d + a
{ b, c, e }

{ b, c, e }
f = b + c
{ b, e, f }

{ b, e, f }
f = f + b
{ e, f }

{ e, f }
d' = e + f
{ d' }

{ e, f }
d' = e - f
{ d' }

{ d' }
g = d'
{ g }

Registers

R₀ R₁ R₂

(spilled)
A Smarter Algorithm

\{ a, b, c, d \}
\[
e = d + a
\]
\{ b, c, e \}

\{ b, c, e \}
\[
f = b + c
\]
\{ b, e, f \}

\{ b, e, f \}
\[
f = f + b
\]
\{ e, f \}

\{ e, f \}
\[
d' = e + f
\]
\{ d' \}

\{ e, f \}
\[
d' = e - f
\]
\{ d' \}

\{ d' \}
\[
g = d'
\]
\{ g \}

Registers

\( R_0 \) \( R_1 \) \( R_2 \)

\( d' \)  (spilled)
Another Example
Another Example
Another Example

Registers

\( R_0 \quad R_1 \quad R_2 \)
Another Example

Registers

R₀   R₁   R₂
Another Example

Registers

\[ R_0 \quad R_1 \quad R_2 \]
Another Example

Registers

\[ R_0 \quad R_1 \quad R_2 \]
Another Example
Another Example

Registers

\[
\begin{array}{ccc}
R_0 & R_1 & R_2 \\
\end{array}
\]
Another Example

Registers

\begin{tabular}{|c|c|c|}
\hline
R_0 & R_1 & R_2 \\
\hline
\end{tabular}
Another Example

Registers

$R_0$  $R_1$  $R_2$
Another Example

Registers

\[ \begin{array}{ccc}
R_0 & R_1 & R_2 \\
\end{array} \]
Another Example

Registers

R₀  R₁  R₂
Another Example

Registers

\begin{tabular}{c|c|c}
\hline
R_0 & R_1 & R_2 \\
\hline
\end{tabular}
Another Example

Registers

\[
\begin{array}{ccc}
R_0 & R_1 & R_2 \\
\end{array}
\]
Another Example
Another Example

Registers

\[ R_0 \quad R_1 \quad R_2 \]
Another Example

```plaintext
Registers
R0  R1  R2
```
Another Example

Registers

\begin{array}{ccc}
R_0 & R_1 & R_2 \\
\end{array}
Another Example

Registers

\[
\begin{array}{ccc}
R_0 & R_1 & R_2 \\
\end{array}
\]
Another Example

Registers

\begin{tabular}{|c|c|c|}
\hline
R_0 & R_1 & R_2 \\
\hline
\end{tabular}
Another Example

Registers

\[
\begin{align*}
R_0 & \quad R_1 & \quad R_2 \\
\end{align*}
\]
Another Example

Registers

R₀  R₁  R₂
Another Example

Registers

\begin{tabular}{c|c|c}
  \hline
  R_0 & R_1 & R_2 \\
  \hline
\end{tabular}
Another Example

Registers

\[
R_0 \quad R_1 \quad R_2
\]
Another Example

Registers

\[ \begin{array}{ccc}
R_0 & R_1 & R_2 \\
\end{array} \]
Another Example

Registers

\begin{tabular}{c|c|c}
  \hline
  R_0 & R_1 & R_2 \\
  \hline
  \end{tabular}
Another Example

Registers

\[ R_0 \quad R_1 \quad R_2 \]
Another Example

Registers

\begin{tabular}{c|c|c}
\hline
\textbf{R} & 0 & 1 & 2 \\
\hline
\end{tabular}
Another Example

Registers

\[
\begin{array}{c|c|c}
R_0 & R_1 & R_2 \\
\end{array}
\]
Another Example

Registers

\begin{align*}
R_0 & \quad R_1 & \quad R_2 \\
\end{align*}
Another Example

Registers

R₀  R₁  R₂
Another Example

Registers

\begin{align*}
\text{R}_0 & \quad \text{R}_1 & \quad \text{R}_2
\end{align*}
Another Example

Registrars

\begin{itemize}
\item \textcolor{cyan}{R_0}
\item \textcolor{cyan}{R_1}
\item \textcolor{cyan}{R_2}
\end{itemize}
Another Example

Registers

\begin{array}{c|c|c}
\text{R}_0 & \text{R}_1 & \text{R}_2 \\
\end{array}
Chaitin's Algorithm

- **Advantages:**
  - For many control-flow graphs, finds an excellent assignment of variables to registers.
  - When distinguishing variables by use, produces a precise RIG.
  - Often used in production compilers like GCC.

- **Disadvantages:**
  - Core approach based on the NP-hard graph coloring problem.
  - Heuristic may produce pathologically worst-case assignments.
Correctness Proof Sketch

- No two variables live at some point are assigned the same register.
  - Forced by graph coloring.
- At any program point each variable is always in one location.
  - Automatic if we assign each variable one register.
  - Requires a few tricks if we separate by use case.
Improvements to the Algorithm

- Choose what to spill intelligently.
  - Use heuristics (least-commonly used, greatest improvement, etc.) to determine what to spill.

- Handle spilling intelligently.
  - When spilling a variable, recompute the RIG based on the spill and use a new coloring to find a register.
Summary of Register Allocation

- Critical step in all optimizing compilers.
- The **linear scan** algorithm uses **live intervals** to greedily assign variables to registers.
  - Often used in JIT compilers due to efficiency.
- **Chaitin's algorithm** uses the **register interference graph** (based on **live ranges**) and **graph coloring** to assign registers.
  - The basis for the technique used in GCC.
Next Time

- **Garbage Collection**
  - Reference Counting
  - Mark-and-Sweep
  - Stop-and-Copy
  - Incremental Collectors