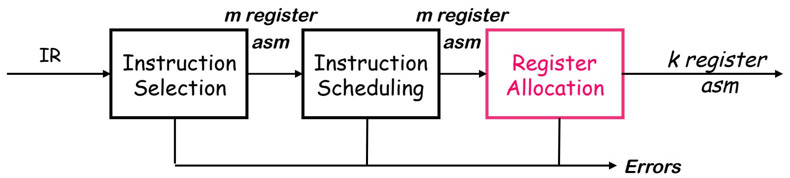
Register Allocation

Note by Baris Aktemur:
Our slides are adapted from Cooper and Torczon's slides that they prepared for COMP 412 at Rice.

Register Allocation

Part of the compiler's back end



Critical properties

- Produce <u>correct</u> code that uses *k* (or fewer) registers
- Minimize added loads and stores
- Minimize space used to hold *spilled values*
- Operate efficiently

  $O(n)$, $O(n \log_2 n)$, maybe $O(n^2)$, but not $O(2^n)$

Global Register Allocation

The Big Picture

Optimal global allocation is NP-Complete, under almost any assumptions.

At each point in the code

1 Determine which values will reside in registers
2 Select a register for each such value

The goal is an allocation that "minimizes" running time

Most modern, global allocators use a graph-coloring paradigm

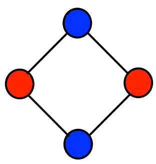
- Build a "conflict graph" or "interference graph"
- Find a *k*-coloring for the graph, or change the code to a nearby problem that it can *k*-color

Graph Coloring (A Background Digression)

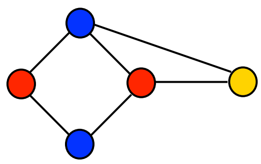
The problem

A graph *G* is said to be *k-colorable* iff the nodes can be labeled with integers 1 … k so that no edge in *G* connects two nodes with the same label

Examples



2-colorable



3-colorable

Each color can be mapped to a distinct physical register

Building the Interference Graph

What is an "interference" ? (or conflict)

- Two values *interfere* if there exists an operation where both are simultaneously live
- If x and y interfere, they cannot occupy the same register

To compute interferences, we must know where values are "live"

We've seen Liveness analysis in the Data Flow Analysis lecture.

Observation on Coloring for Register Allocation

- Suppose you have *k* registers—look for a *k* coloring
- Any vertex *n* that has fewer than *k* neighbors in the interference graph ($n^\circ < k$) can always be colored!
  - Pick any color not used by its neighbors — there ***must*** be one

Chaitin's Algorithm

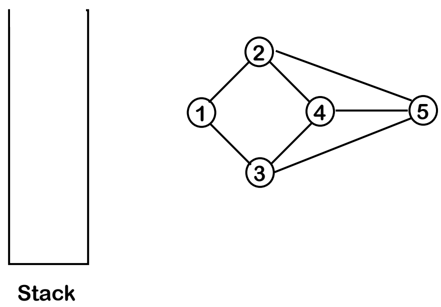
1. While ∃ vertices with < k neighbors in G_I
   > Pick any vertex n such that $n^{\circ} < k$ and put it on the stack
   > Remove that vertex and all edges incident to it from G_I

   Lowers degree of n's neighbors

2. If G_I *is non-empty* (all vertices have k or more neighbors) then:
   > Pick a vertex n (using some heuristic) and spill the live range associated with n
   > Remove vertex n from G_I, along with all edges incident to it and put it on the "spill list"
   > If this causes some vertex in G_I to have fewer than k neighbors, then go to step 1; otherwise, repeat step 2
3. If the spill list is not empty, insert spill code, then rebuild the interference graph and try to allocate, again
4. Otherwise, successively pop vertices off the stack and color them in the lowest color not used by some neighbor

Chaitin's Algorithm in Practice

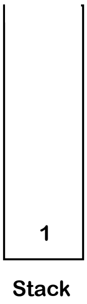
3 Registers

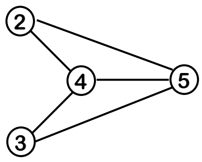


1 is the only node with degree < 3

Chaitin's Algorithm in Practice

3 Registers

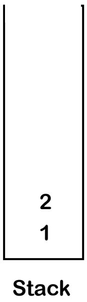


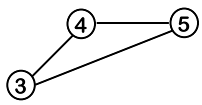


Now, 2 & 3 have degree < 3

Chaitin's Algorithm in Practice

3 Registers

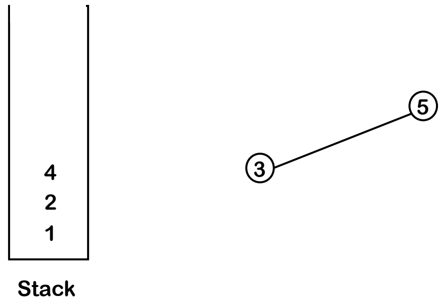




Now all nodes have degree < 3

Chaitin's Algorithm in Practice

3 Registers



Chaitin's Algorithm in Practice

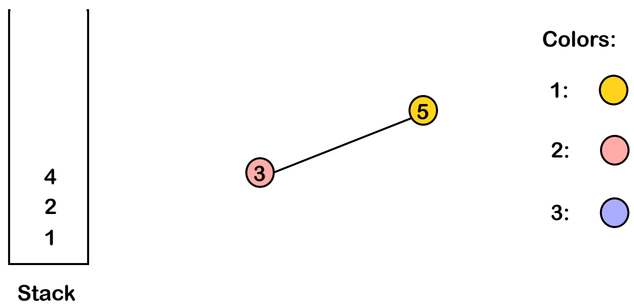
3 Registers

Chaitin's Algorithm in Practice

3 Registers

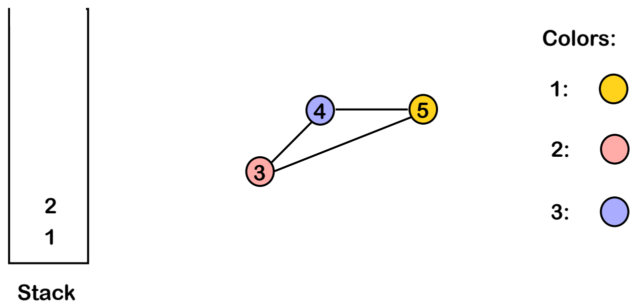
Chaitin's Algorithm in Practice

3 Registers



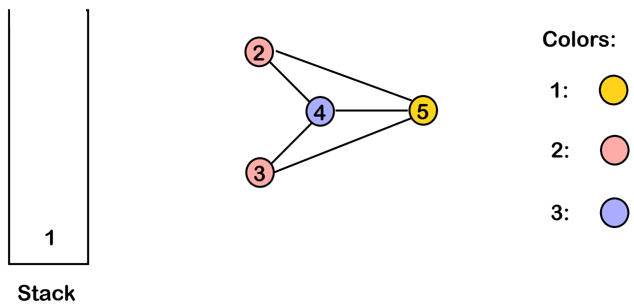
Chaitin's Algorithm in Practice

3 Registers



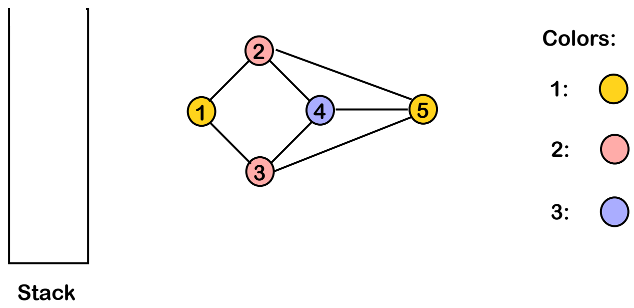
Chaitin's Algorithm in Practice

3 Registers



Chaitin's Algorithm in Practice

3 Registers

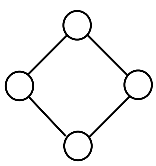


Improvement in Coloring Scheme

Optimistic Coloring

- If Chaitin's algorithm reaches a state where every node has *k* or more neighbors, it chooses a node to spill.
- Briggs said, take that same node and push it on the stack
  - When you pop it off, a color might be available for it!

2 Registers:



Chaitin's algorithm immediately spills one of these nodes

  - For example, a node *n* might have *k+2* neighbors, but those neighbors might only use 3 (<*k*) colors
    - → Degree is a *loose upper bound* on colorability

Briggs et al, PLDI 89 (Also, TOPLAS 1994)

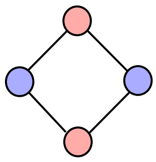
Improvement in Coloring Scheme

Optimistic Coloring

- If Chaitin's algorithm reaches a state where every node has *k* or more neighbors, it chooses a node to spill.
- Briggs said, take that same node and push it on the stack
  - When you pop it off, a color might be available for it!

2 Registers:

2-Colorable



Briggs algorithm finds an available color

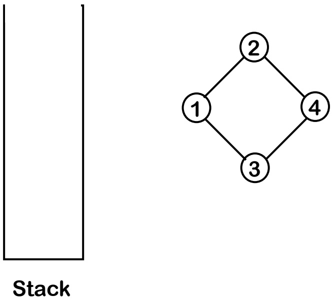
  - For example, a node *n* might have *k+2* neighbors, but those neighbors might only use just one color (*or any number < k*)
    - → Degree is a *loose upper bound* on colorability

Chaitin-Briggs Algorithm

1. While ∃ vertices with < *k* neighbors in G_I
   - > Pick any vertex *n* such that $n^\circ < k$ and put it on the stack
   - > Remove that vertex and all edges incident to it from G_I
     - → This action often creates vertices with fewer than *k* neighbors
2. If G_I is non-empty (all vertices have *k* or more neighbors) then:
   - > Pick a vertex *n* (using some heuristic condition), push *n* on the stack and remove *n* from G_I, along with all edges incident to it
   - > If this causes some vertex in G_I to have fewer than *k* neighbors, then go to step 1; otherwise, repeat step 2
3. Successively pop vertices off the stack and color them in the lowest color not used by some neighbor
   - > If some vertex cannot be colored, then pick an uncolored vertex to spill, spill it, and restart at step 1

Chaitin-Briggs in Practice

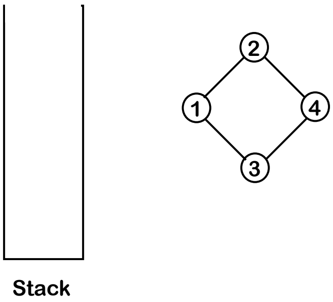
2 Registers



No node has degree < 2
- Chaitin would spill a node
- Briggs picks the same node & stacks it

Chaitin-Briggs in Practice

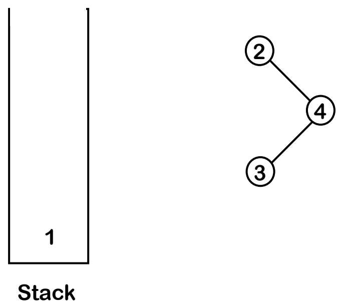
2 Registers



Pick a node, say 1

Chaitin-Briggs in Practice

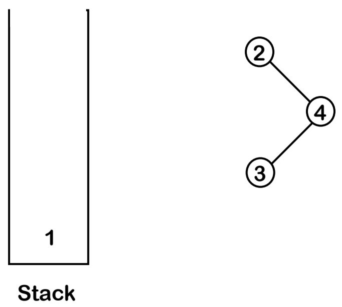
2 Registers



Pick a node, say 1

Chaitin-Briggs in Practice

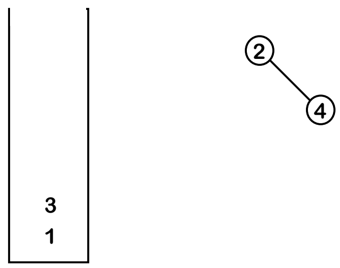
2 Registers



Now, both 2 & 3 have degree < 2
Pick one, say 3

Chaitin-Briggs in Practice

2 Registers

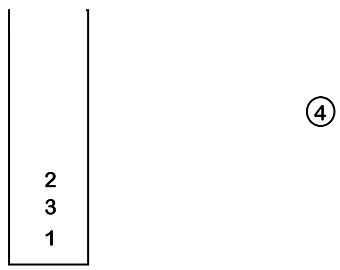


Stack

Both 2 & 4 have degree < 2.
Take them in order 2, then 4.

Chaitin-Briggs in Practice

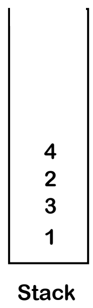
2 Registers



Stack

Chaitin-Briggs in Practice

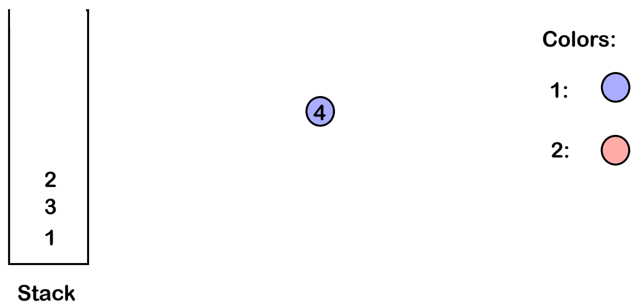
2 Registers



Now, rebuild the graph

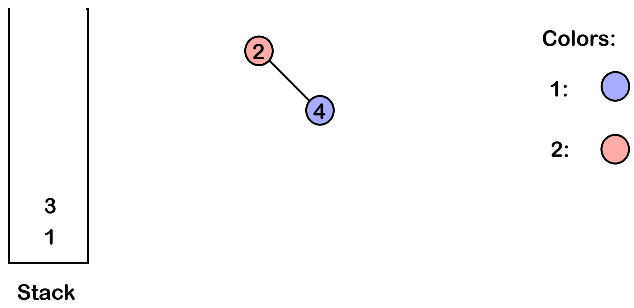
Chaitin-Briggs in Practice

2 Registers



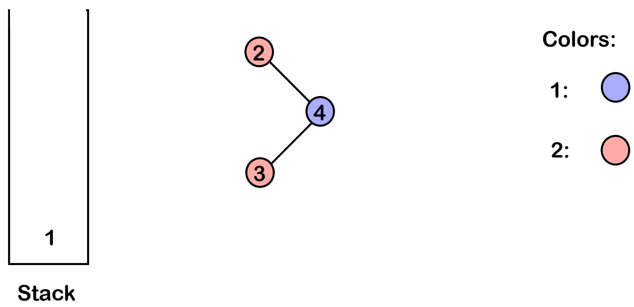
Chaitin-Briggs in Practice

2 Registers



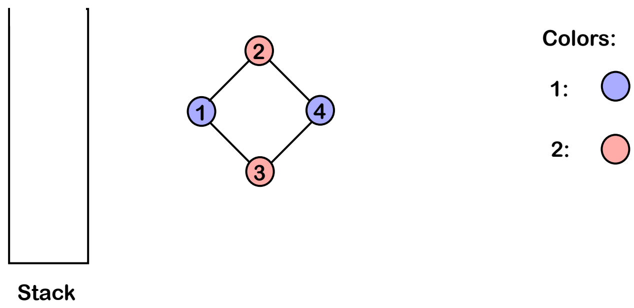
Chaitin-Briggs in Practice

2 Registers



Chaitin-Briggs in Practice

2 Registers



Linear Scan Allocation

Coloring allocators are often viewed as too expensive for use in JIT environments, where compile time occurs at runtime

Linear scan allocators use an approximate interference graph

Algorithm does allocation in a "linear" scan of the graph

Linear scan produces faster, albeit less precise, allocations

Linear scan allocators hit a different point on the curve of cost versus performance

 Sun's HotSpot server compiler uses a complete Chaitin-Briggs allocator.