CS 4120
Introduction to Compilers

Ross Tate
Cornell University

Lecture 29: Register Allocation

Review

- Want to replace all variables (including temporaries) with some fixed set of registers if possible
- First: need to know which variables are *live* after each instruction
- Two simultaneously live variables cannot be allocated to same register

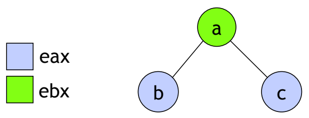
Register allocation

- For every node *n* in CFG now have *out*[*n*] : which variables (temporaries) are live on exit from node.
- If two variables are in same live set, can't be allocated to the same register – they *interfere* with each other
- How do we assign registers to variables?

Inference Graph

- Nodes of graph: variables
- Edges connect all variables that *interfere* with each other
- Register assignment is graph coloring

```
              a
b = a + 2;    a,b
c = b*b;      a,c
b = c + 1;    a,b
return b*a;
```

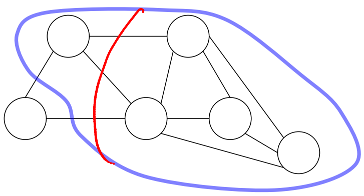


Graph Coloring

- Questions:
  - Can we efficiently find a coloring of the graph whenever possible?
  - Can we efficiently find the optimum coloring of the graph?
  - How can we choose registers to avoid mov instructions?
  - What do we do when there aren't enough colors (registers) to color the graph?

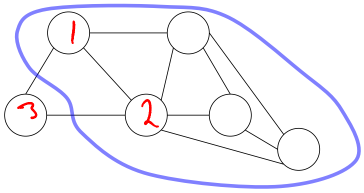
Coloring a Graph

- Kempe's algorithm [1879] for finding a K-coloring of a graph: (Assume K=3)
- Step 1: find some node with at most K-1 edges and cut it out of graph (simplify)



Kempe's Algorithm

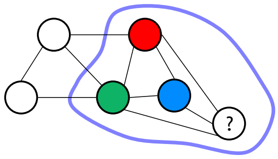
- Once coloring is found for simplified graph, selected node can be colored using free color
- Step 2: simplify until graph contain no nodes, unwind adding nodes back & assigning colors



7

Failure of heuristic

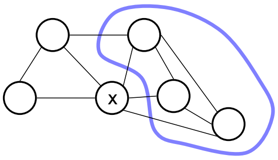
- Failure: reduces to a graph in which every node has at least K neighbors
- May happen even if graph is colorable in K!
- Finding K-coloring is NP-hard problem (requires search)



8

Spilling

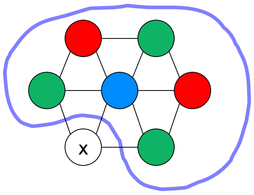
- Once all nodes have K or more neighbors, pick a node and mark it for ***spilling*** (storage on stack). Remove it from graph, continue as before
- Try to pick node not used much, not in inner loop



9

Optimistic Coloring

- Spilled node may be K-colorable; when assigning colors, try to color it and only spill if necessary.
- If not colorable, record this node as one to be spilled, assign it a stack location and keep coloring



10

Accessing spilled variables

- Need to generate additional instructions to get spilled variables out of stack and back in again
- Naive approach: always keep extra registers handy for shuttling data in and out. Problem: uses up 3 registers!
- Better approach: rewrite code introducing a new temporary, rerun liveness analysis and register allocation

11

Rewriting code

```
add t1, t2
```

- Suppose that t2 is selected for spilling and assigned to stack location [ebp-24]
- Invent new variable t35 *for just this instruction*, rewrite:

```
mov t35, [ebp - 24]
add t1, t35
```

- Advantage: t35 doesn't interfere with as much as t2 did. Now rerun algorithm; fewer or no variables will spill.

12

Precolored nodes

- Some variables are pre-assigned to registers
- mul instruction has
  use(*n*) = eax, *def* (*n*) = { eax, edx }
- call instruction kills caller-save regs:
  def (*n*) = { eax, ecx, edx }
- To properly allocate registers, treat these register uses as special temporary variables and enter into interference graph as *precolored nodes*

Simplifying graph with precolored nodes

- Can't simplify graph by removing a pre-colored node
- Precolored nodes: starting point of coloring process
- Once simplified graph is all colored nodes, add other nodes back in and color them

Optimizing mov instructions

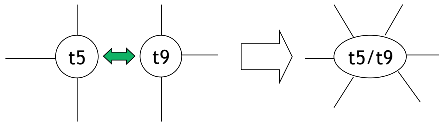
- Code generation produces a lot of extra mov instructions

  mov t5, t9

- If we can assign t5 and t9 to same register, we can get rid of the mov
- Idea: if t5 and t9 are not connected in inference graph, *coalesce* them into a single variable. mov will be redundant.

Coalescing

- Problem: coalescing two nodes can make the graph uncolorable
- High-degree nodes can make graph harder to color, even impossible
- Avoid creation of high-degree (>K) nodes *(conservative coalescing)*



Simplification + Coalescing

- Start by simplifying as much as possible without removing nodes that are *either* the source or destination of a mov *(move-related nodes)*
- Coalesce some pair of move-related nodes as long as low-degree node results; delete corresponding mov instruction(s)
- If can neither simplify nor coalesce, take a move-related pair and ***freeze*** the mov instruction, do not consider nodes move-related

High-level algorithm