Register Allocation: Coloring

- Register Allocation and Coloring
- Building the Interference Graph
- Register Coloring

Register Allocation

- Problem
  - Allocation of variables (pseudo registers) to hardware registers in a procedure
- Important Optimization
  - Directly reduces execution time since register accesses are faster than memory accesses
    - Gets more important as the processor speed grows much faster than memory accesses

Terminology

- Allocation
  - Decision to keep a pseudo register in a hardware register
- Spilling
  - A pseudo register is spilled to memory, if not kept in a hardware register
- Assignment
  - Decision to keep a pseudo register in a **specific** hardware register

What are the Problems?

What is the minimum number of registers needed to avoid spill?

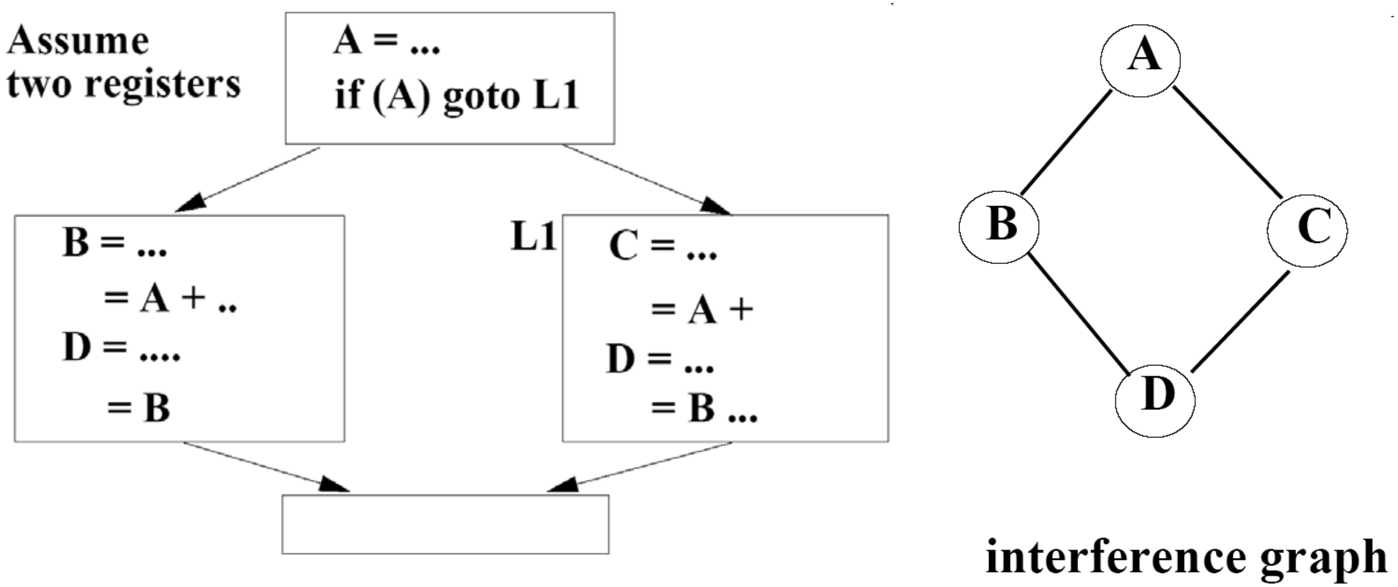
- Given *n* registers in a machine, is spilling really necessary?
- Find an assignment for pseudo registers if we can allocate all
- If there are not enough registers in the machine, however, how do we spill to the memory, with minimal costs?

Advanced issues

- How can we remove copies as well via register allocation ?

Interference

- When cannot we allocate the same register to two different pseudo registers? when they interfere
- Two pseudo registers interfere if at some point in the program they are live simultaneously. For example,



Abstraction for Interference & Allocation

Interference graph: an undirected graph where

- Nodes: pseudo registers
- There is an edge between two nodes if their corresponding pseudo register interfere

Register allocation on interference graph is modeled by coloring nodes in the graph

- Colors are hardware registers
- We cannot color two nodes with the same color if they are adjacent (connected by an edge)

Coloring Interference Graph

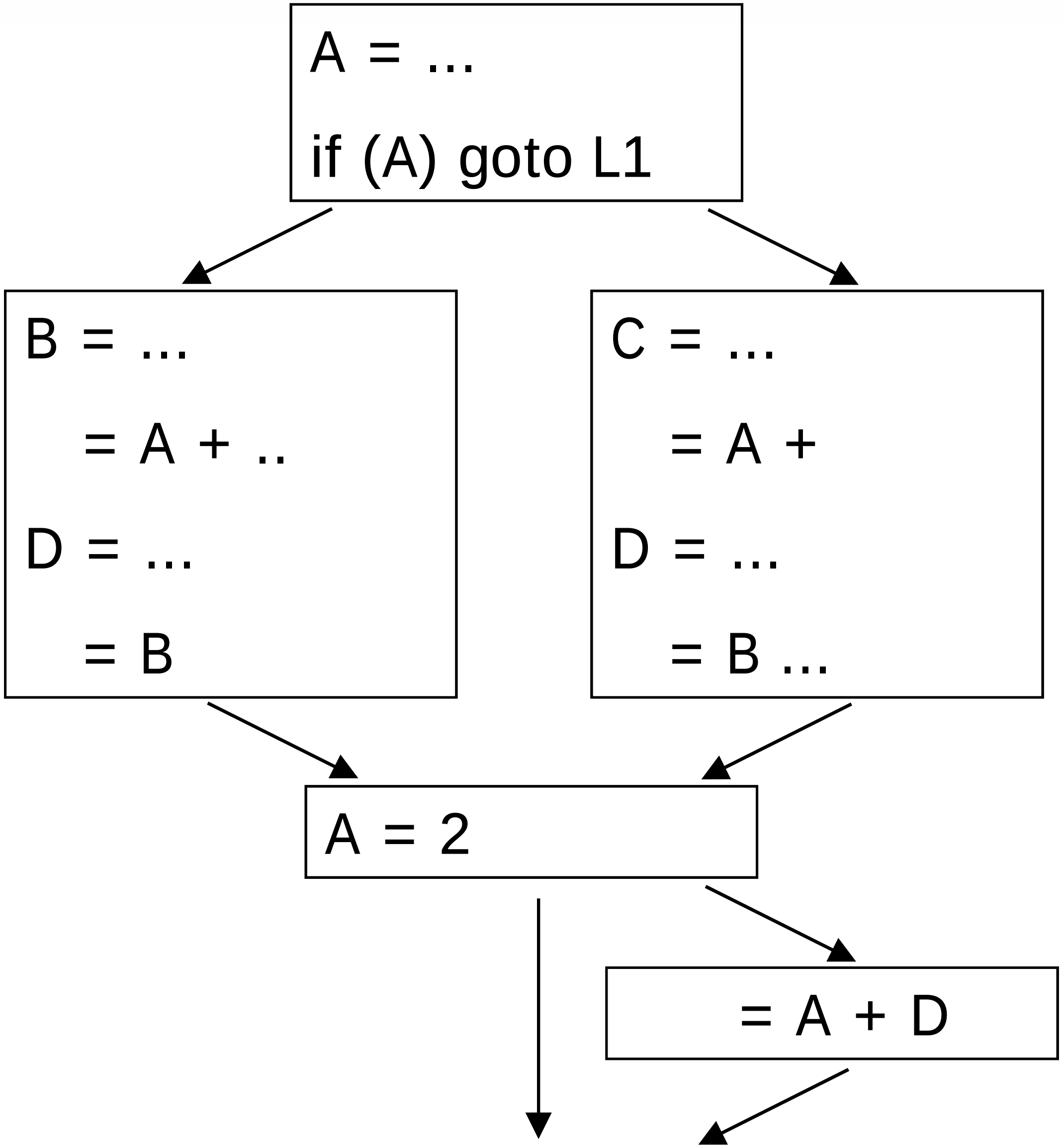
- A graph is n-colorable if
  - each node in the graph can be colored with one of *n* colors such that no two adjacent nodes are assigned same color
- Assigning *n* registers without spilling
  = coloring with *n* colors
- Is spilling necessary? = Is the graph *n*- colorable?
- Determining if a graph is *n*-colorable is NP-complete

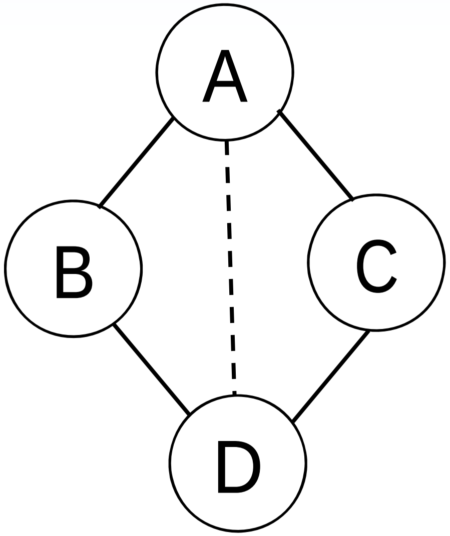
Building Interference Graph

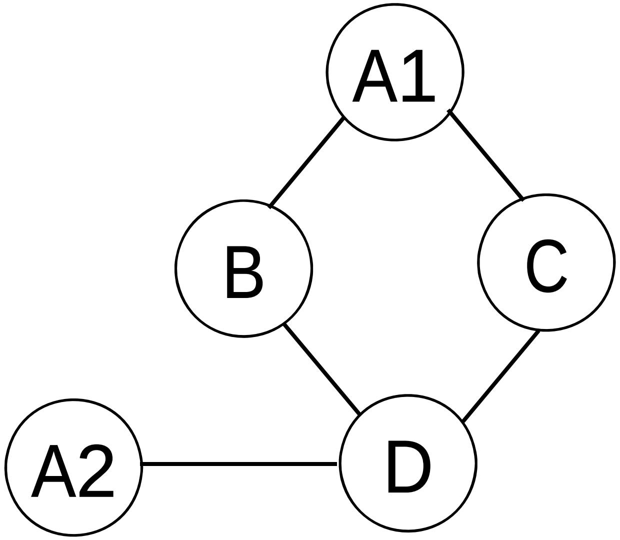
Two issues

- How to define nodes?
  - Pseudo registers can be nodes, but we need to refine them further using the idea of live ranges
- How to find edges?
  - Two nodes that are simultaneously live at some point of program are not necessarily interfering

Nodes in an Interference Graph

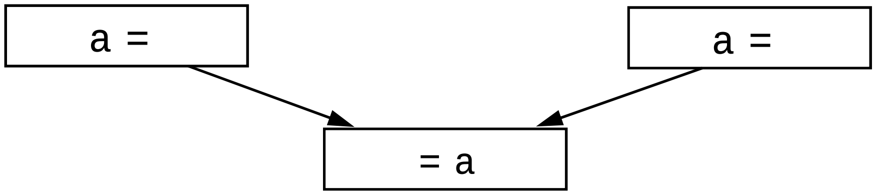






Live Ranges and Merged Live Ranges

- Motivation: Create an interference graph that is easier to color
  - Eliminate interference in a variable's dead zones
  - Increase flexibility in allocation:
    can allocate same variable to different registers
- A Live range consists of a definition and all the points in a program (e.g., end of an instruction) where that definition is live
  - How to compute a live range?
    a point $p \in$ live range of a definition d (a=b+c)
    - iff (1) d must reach p and (2) a must be live
- Two overlapping live ranges for the same variable must be merged



Merging Live Range

- Merging definitions into equivalent classes
  - Start by putting each definition in a different equivalent class
  - For each point in a program,
    - If variable is live and there are multiple reaching definitions for the variable
    - Merge the equivalence classes of all such definitions into a one equivalence class
- From now on, refer to merged live range simply as live ranges

Edges of Interference Graph

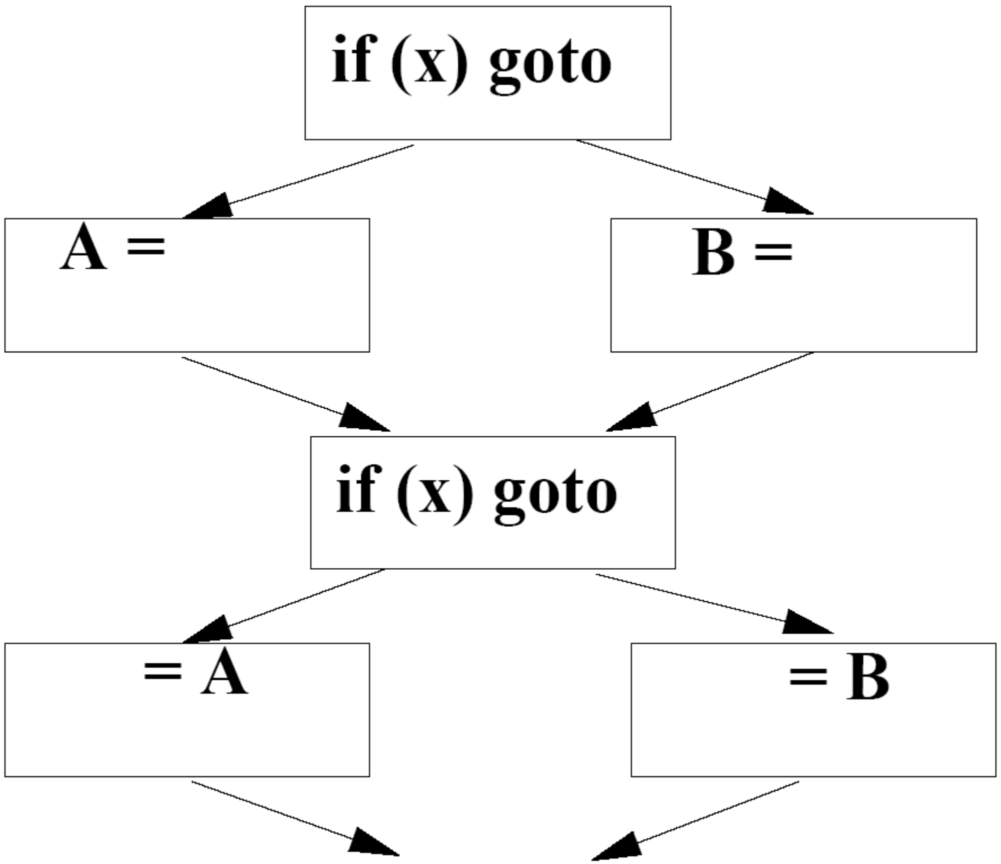
- Intuitive Algorithm
  - Two live ranges may interfere if they overlap at some point in the program
  - Algorithm: At each point in the program, enter an edge for every pair of live ranges at that point

- Optimized Algorithm

```
For each instruction I
      Let x be the live range of definition at instruction I
      For each live range y present at end of instruction I
            insert an edge between x and y
```

Faster and Better Quality

Example



intuitive algorithm

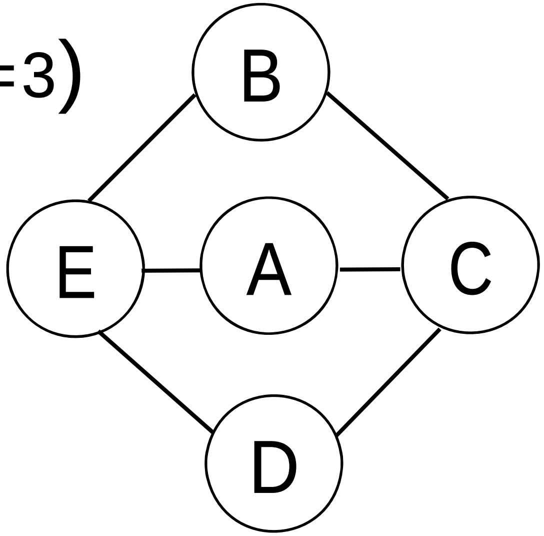
optimized algorithm

Coloring the Graph

- Use heuristics to try to find an *n*-coloring
  - Successful: Colorable and we have an assignment
  - Failure : Graph not colorable, or graph is colorable but it is too expensive to color
- Observation
  - A node with degree $< n$ can always be colored successfully, given its neighbors' colors
  - What about a node with degree $= n$?
  - What about a node with degree $> n$?
  - When are we sure that the graph is not colorable?

Coloring Algorithm

- Algorithm
  - Iterate until stuck or done
    - Pick any node with degree < n
    - Remove the node and its edges from the graph
  - If done (no nodes left)
    - Reverse process and add colors
- Example (n=3)



Observation:

* Degree of a node may drop in iteration
* We should avoid making arbitrary decisions that make coloring fail

What does coloring accomplish?

* Done: colorable, also obtained an assignment
* Stuck: colorable or not?