Register Allocation

Reasons

- The Translate Canon and Code-gen phases of the compiler assume that
  - there are an infinite number of registers to hold temporary values and
  - that MOVE instructions cost nothing.

The Job of Register Allocation

- To assign the many temporaries to a small number of machine languages .
- Where possible, to assign the source and destination of a MOVE to the same registers so that the MOVE can be deleted.

- From the examination of the control and data-flow graph, we drive an *interference graph.*
  - *Each node represents a temporary value;*
  - *each edge(t1,t2) indicates a pair of temporaries that cannot assign to the same register.*
  - *t1,t2 are live at the same time.*
  - *Edge express other constrains;*

- Next, We color the interference graph.
  - *Use as few colors as possible, but no pair of nodes connected by an edge may be assigned the same color.*
    - *If our target machine has K registers, and we can K-color the graph, then the coloring is a valid register assignment for the interference graph.*
    - *If there is no K-coloring, we will have to keep some of our variables and temporaries in memory instead of registers: this is called spilling.*

COLORING BY SIMPLIFICATION

- The principal phases
  - Build
  - Simplify
  - Spill
  - Select

- Build
  - Construct the interference graph. We use data-flow analysis to compute the set of registers that are simultaneously live at each program point, and we add an edge to the graph for each pair of registers in the set.

- Simplify
  - –We color the graph using a simple heuristic.

Suppose:

Graph G contains a node m with fewer than K neighbors, where K is the number of registers on the machine.

Let **G'=G-{m}**

If G' can be colored, then so can G.

- Each such simplification will decrease the degrees of other nodes, leading to more opportunity for simplification.

- Spill
  - Suppose at some point during simplification the graph G has nodes only of significant degree, that is, nodes of degree >=k. then the simplify heuristic fails and we mark some node for spilling.
  - An optimistic approximation to the effect of spilling is that the spilled node does not interfere with other nodes remaining in the graph.

- Select
  - We assign colors to nodes in the graph.
  - Starting with the empty graph, we re-build the original graph by repeatedly adding a node from the top of the stack.
  - When we add a node to the graph, there must be a color for it.
  - *Potential spill*
    - ***actual spill***
    - ***optimistic coloring.***

- Start over
  - if the select phase is unable to find a color for some node(s), then the program must be **rewritten** to fetch them from memory just before each use, and store them back after each definition.
  - Thus , a spilled temporary will turn into several new temporaries with tiny live ranges.

Line-in: k j

g:=mem[j+12]

h:=k-1

f:=g*h

e:=mem[j+8]

m:=mem[j+16]

b:=mem[f]

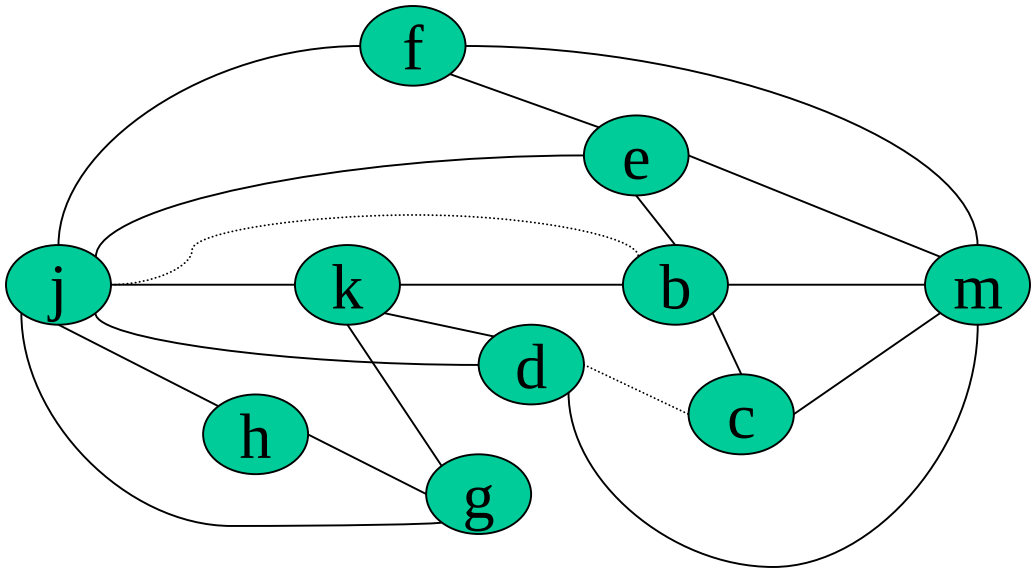
c:=e+8

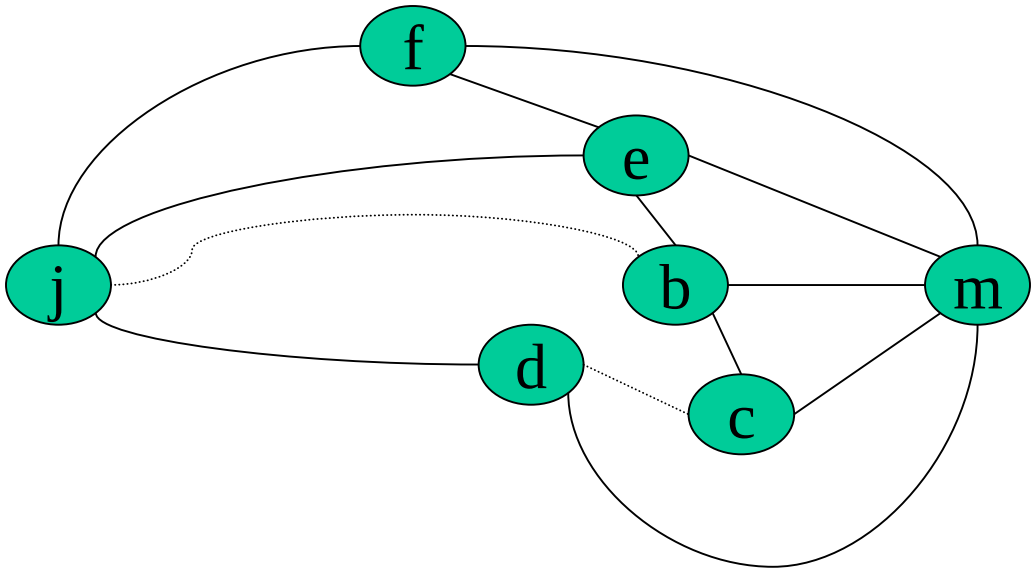
d:=c

k:=m+4

j:=b

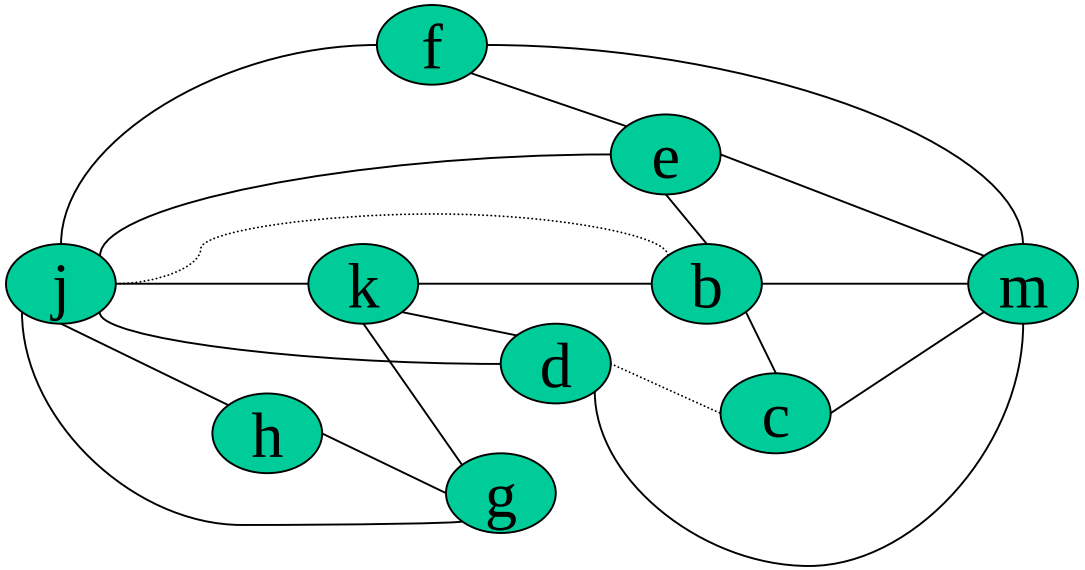
Live-out: d k j





After removal of h, g and k

m	1
c	3
b	2
f	2
e	4
j	3
d	4
k	1
h	2
g	4



(a)stack (b)assignment

Simplification stack, and a possible coloring

COALESCING

- If there is no edge in the interference graph between the source and destination of a move instruction, then the move can be eliminated.
- The source and destination nodes are coalesced into a new node whose edges are union of those of the nodes being replaced.

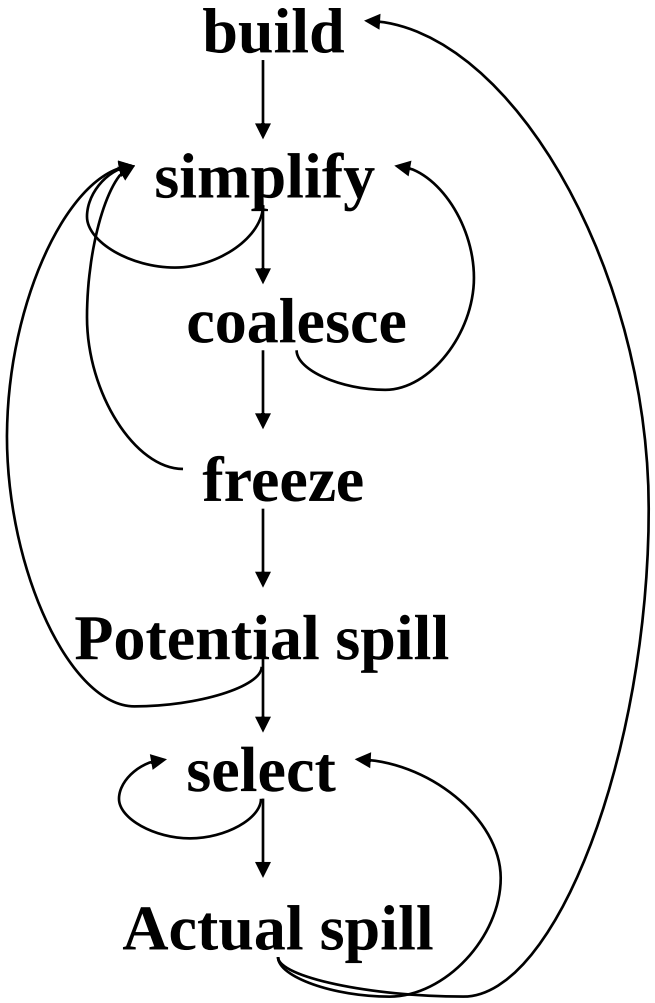
- If some nodes are precolored , they cannot be spilled.
- If a temporary interferes with K precolored nodes, then the temporary must be spilled.
- But there is no register into which it can be fetched back for computation ! We say such a graph is un-colorable.

Conservative coalescing

- If the node being coalesced has fewer than K neighbors of significant degree, then coalescing is guaranteed not to turn a K-colorable graph into non-K-colorable graph.

The phase of a register allocator with coalescing

- Build
  - Construct the interference graph and categorize each node as either move-related or non-move-related.
  - Move-related node is one that is the either the source or destination of move instruction.
- Simplify
  - One at a time, remove non-move-related nodes of low (<K) degree from the graph.

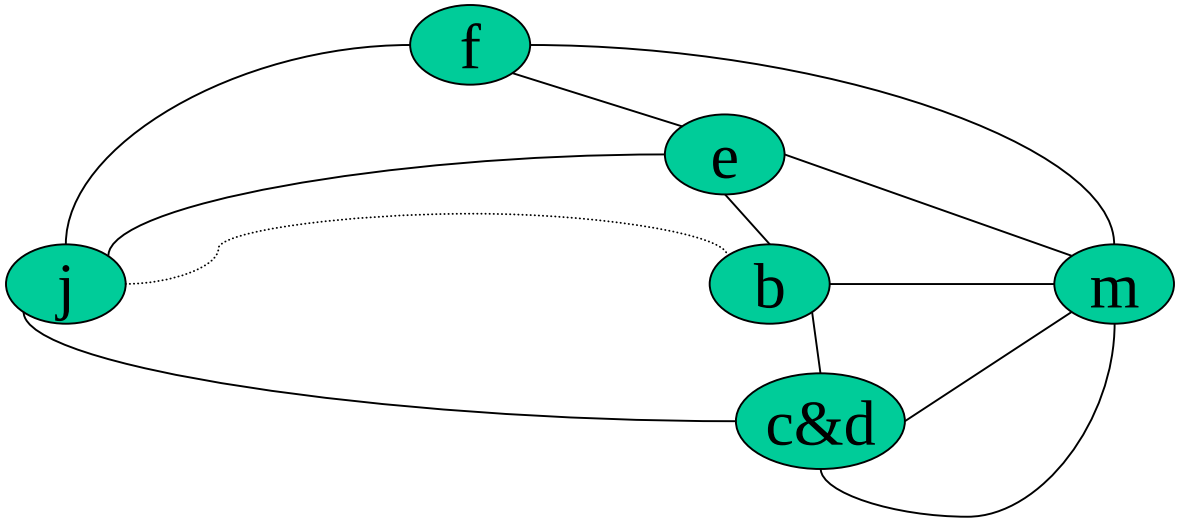


Graph coloring with coalescing

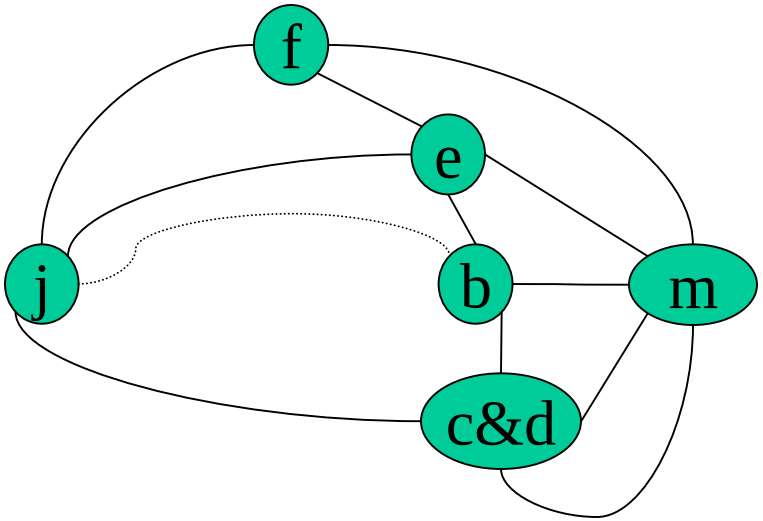
- Coalesce
  - Perform conservative coalescing on the reduced graph.
  - After two nodes have been coalesced and the move instruction deleted, if the resulting node is no longer move-related it will be available for the next round of simplification.
  - Simplify and coalesce are repeated until only significant-degree or move-related nodes remain.

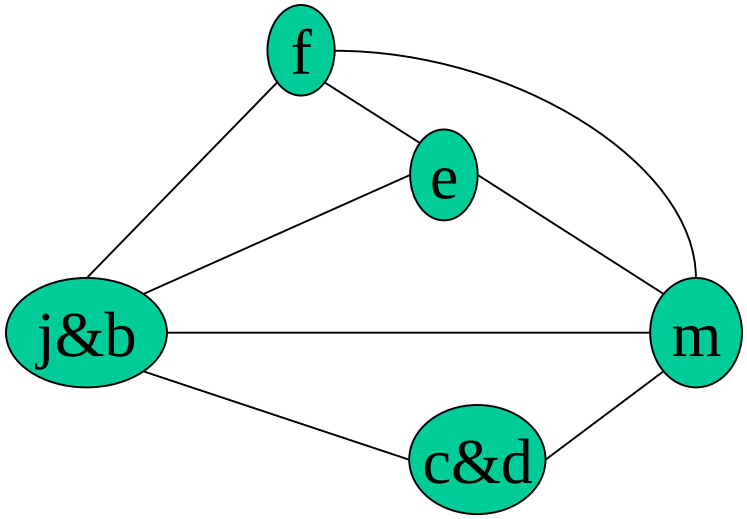
- Freeze
  - If neither simplify nor coalesce applies, we look for a move-related node of low degree.
  - We freeze the moves in which this node is involved: that is ,we give up hope of coalescing those moves.
  - Now, simplify and coalesce are resumed.

- Spill
  - If there are no low-degree nodes, we select a high-degree node for potential spilling and push it on the stack.
- Select
  - Pop the entire stack, assigning colors.



- **Nodes b,c,d and j are only move-related nodes.**
- **In Simplify phase,**
  working list :nodes g,h,f
- **After removal of nodes g h and k.**
- **After coalescing c & d**

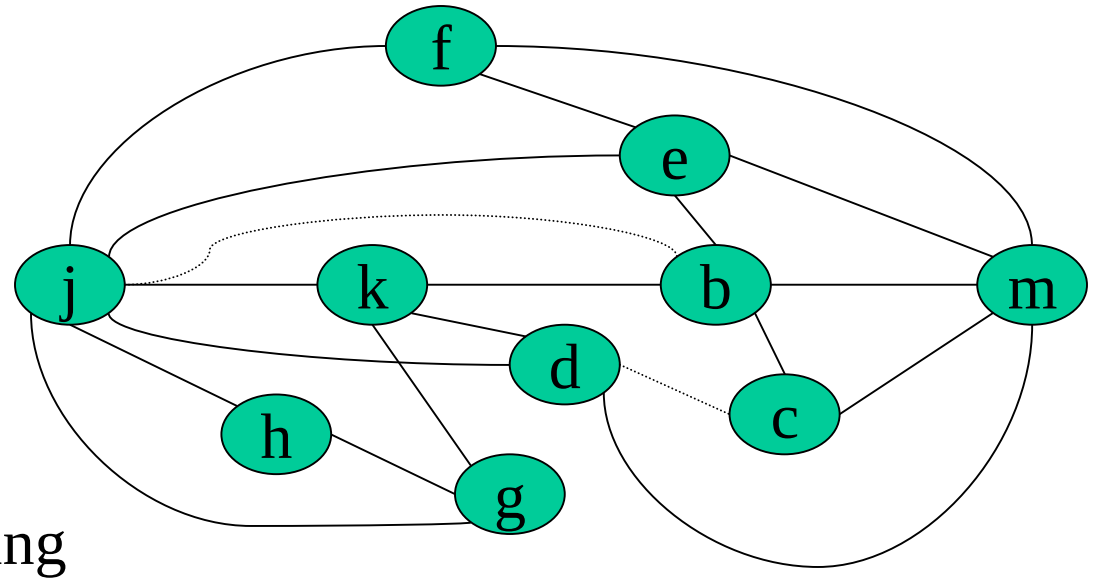




•Nodes b and j are adjacent to two neighbors of significant degree, namely m and e.
•Coalescing b and j
•No more move-related nodes, and therefore no more coalescing is possible.

After coalescing b and j

e	1
m	2
f	3
j&b	4
c&d	1
k	2
h	2
g	1
(a)stack	(b)coloring



A coloring with coalescing

CALLER-SAVE AND CALLEE-SAVE REGISTERS

- Caller-save register
  - A local variable or compiler temporary that is not live across any procedure call should be allocated to a caller-save register.
  - No saving and restoring of the register.
- Callee-save register
  - Any variable that is live across several procedure calls should be kept in a callee-save register.
  - Only one save/restore