# Register Allocation

Compiler Design

CSE 504


1. Preliminaries
2. Graph Coloring
3. Spilling



Last modifled: Wed Feb 04 2015 at 12:58:26 EST
Version: 1.4 18:44:24 2015/03/06
Compiled at 21:57 on 2016/04/23






## Final Code Generation

- Intermediate Code assumed a machine with few physical limitations.
- For final code generation, intermediate instructions must be mapped to machine instructions
  - This is called instruction selection, and generally done using a dynamic programming algorithm (tiling).
- Variables may be allocated in registers instead of memory, mapping multiple variables without overlapping lifetimes to the same register/location.
  - This is the task of register allocation.

# Lifetime of Variables

- A variable is *live*
  - from the program point where it is assigned a value
  - to the point where its value is no longer used.
- The extent of a variable's lifetime can be found by a *static program analysis*, called **liveness analysis**.





# Liveness Analysis

- $use(s)$: the set of variables used in statement $s$ (e.g. rhs of an assignment statement).
- $def(s)$: the set of variables defined in statement $s$ (e.g. lhs of an assignment statement).
- $succ(s)$: the set of statements that succeed $s$.
- $out(s)$: the set of variables that are live immediately after $s$.
- $in(s)$: the set of variables that are live immediately before $s$.

$$\begin{aligned} in(s) &= use(s) \cup (out(s) - def(s)) \\ out(s) &= \bigcup_{t \in succ(s)} in(t) \end{aligned}$$

# Liveness Analysis: an example

$$in(s) = use(s) \cup (out(s) - def(s)) \qquad out(s) = \bigcup_{t \in succ(s)} in(t)$$

| Stmt | Def | Use | In | Out |
|---|---|---|---|---|
| `g = M[j+12]` | {g} | {j} | {j, k} | {g, j, k} |
| `h = k - 1` | {h} | {k} | {g, j, k} | {g, h, j} |
| `f = g * h` | {f} | {g,h} | {g, h, j} | {f, j} |
| `e = M[j+8]` | {e} | {j} | {f, j} | {e, f, j} |
| `m = M[j+16]` | {m} | {j} | {e, f, j} | {e, f, m} |
| `b = M[f]` | {b} | {f} | {e, f, m} | {b, e, m} |
| `c = e + 8` | {c} | {e} | {b, e, m} | {b, c, m} |
| `d = c` | {d} | {c} | {b, c, m} | {b, d, m} |
| `k = m + 4` | {k} | {m} | {b, d, m} | {b, d, k} |
| `j = b` | {j} | {b} | {b, d, k} | {d, j, k} |





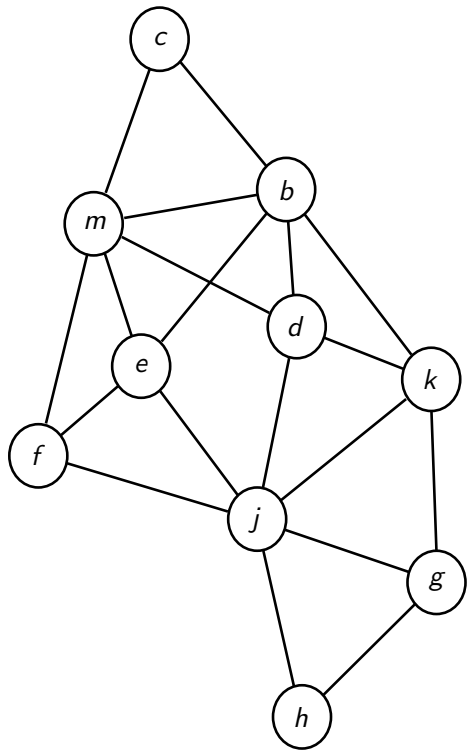
# Liveness Information

- If two variables are live at the same program point, then they cannot be both allocated to the same register/location.
- $v_1$ and $v_2$ *interfere* if $\{v_1, v_2\} \subseteq in(s)$ or $out(s)$ for some statement $s$.
- We build an *interference graph* with
  - variables as vertices, and
  - edge $(v_1, v_2)$ if $v_1$ and $v_2$ interfere.

# Interference Graph Example

| Stmt | In |
|---|---|
| `g = M[j+12]` | {j, k} |
| `h = k - 1` | {g, j, k} |
| `f = g * h` | {g, h, j} |
| `e = M[j+8]` | {f, j} |
| `m = M[j+16]` | {e, f, j} |
| `b = M[f]` | {e, f, m} |
| `c = e + 8` | {b, e, m} |
| `d = c` | {b, c, m} |
| `k = m + 4` | {b, d, m} |
| `j = b` | {b, d, k} |
| `end` | {d, j, k} |

From Appel's "Modern Compiler Implementation in Java" book.








# Register Allocation as Graph Coloring

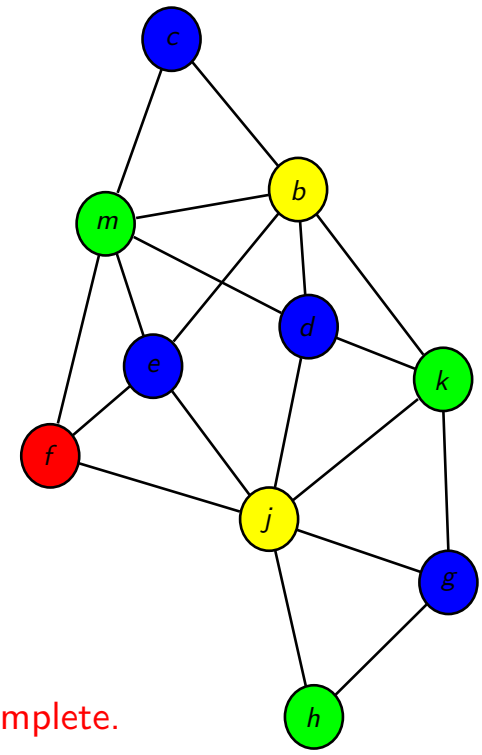
- **Graph Coloring:** an assignment of colors (from a finite set) to vertices such that

  no two adjacent vertices have the same color.

- **Register Allocation:** an assignment of registers (colors) to variables (vertices)

  no two adjacent vertices (i.e variables live at same time)

  have the same register (color).




Unfortunately, graph coloring is NP-complete.

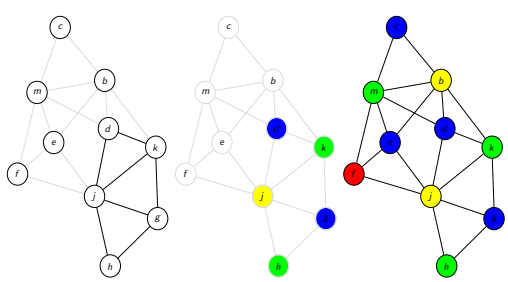
# Graph Coloring Heuristic 1: Simplify & Select

```
procedure K-Color(G, K)
    stack = empty
    while ∃v ∈ G with degree < K do                          ▷ Simplify
        E = set of edges incident on v;    Remove v and E from G
        push(v, E)
    end while
    if G is empty then                                       ▷ Select
        while stack is not empty do
            (v, E) = pop();   Add v and E to G
            color(v) = c, such that c ∉ {color(u)|(u, v) ∈ G}
                                        ▷ Note |E| < K, so there is such a c
        end while
    else
        Exit with failure
    end if
end procedure
```



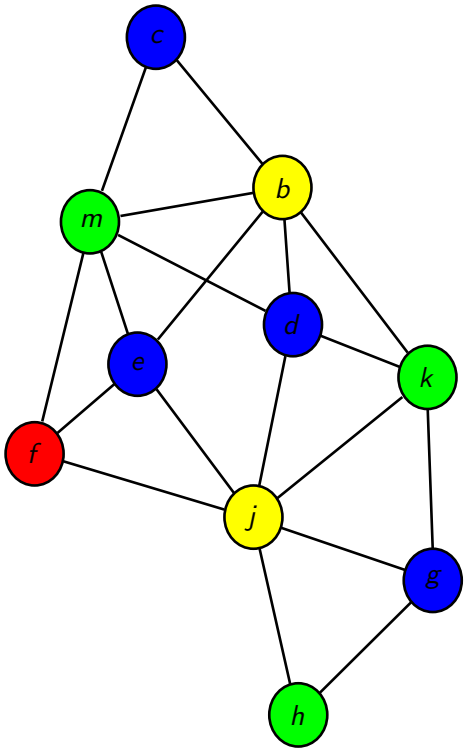


# Graph Coloring Heuristic 1: Example $K = 4$



| Stack: |
|---|
| f |
| e |
| m |
| c |
| b |
| d |
| j |
| k |
| h |
| g |

# Code With Registers




| Stmt | With Regs |
|---|---|
| `g = M[j+12]` | `r0 = M[r2+12]` |
| `h = k - 1` | `r1 = r1 - 1` |
| `f = g * h` | `r3 = r0 * r1` |
| `e = M[j+8]` | `r0 = M[r2+8]` |
| `m = M[j+16]` | `r1 = M[r2+16]` |
| `b = M[f]` | `r2 = M[r3]` |
| `c = e + 8` | `r0 = r0 + 8` |
| `d = c` | `r0 = r0` |
| `k = m + 4` | `r1 = r1 + 4` |
| `j = b` | `r2 = r2` |

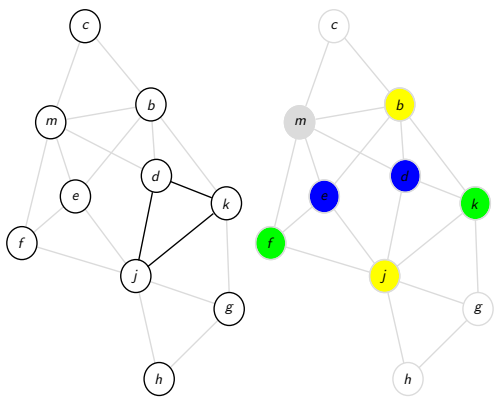




# Graph Coloring Heuristic 2: Simplify, Spill & Select

**procedure** K-Color($G, K$)
  *stack* = *empty*
  **while** $G$ is not *empty* **do**
    **if** $\exists v \in G$ with degree $< K$ **then** ▷ Simplify
      Select such a $v$
    **else** ▷ Spill
      Select a $v$ based on spill priority
    **end if**
    $E$ = set of edges incident on $v$; Remove $v$ and $E$ from $G$
    push($v$, $E$)
  **end while**
  **while** *stack* is not *empty* **do** ▷ Select
    $(v, E)$ = pop(); Add $v$ and $E$ to $G$
    $C = \{\text{color}(u) | (u, v) \in G\}$
    **if** $|C| < K$ **then**
      color($v$) = $c$, such that $c \notin C$
    **else**
      Exit with failure
    **end if**
  **end while**
**end procedure**

# Graph Coloring Heuristic 2: Example $K = 3$




| Stack: |
|---|
| c |
| h |
| g |
| m |
| f |
| e |
| b |
| d |
| j |
| k |
| |

- `m` is selected for potentially spilling: it is rarely used, and has high degree.
- `m` needs to be spilled: stored and restored from memory.



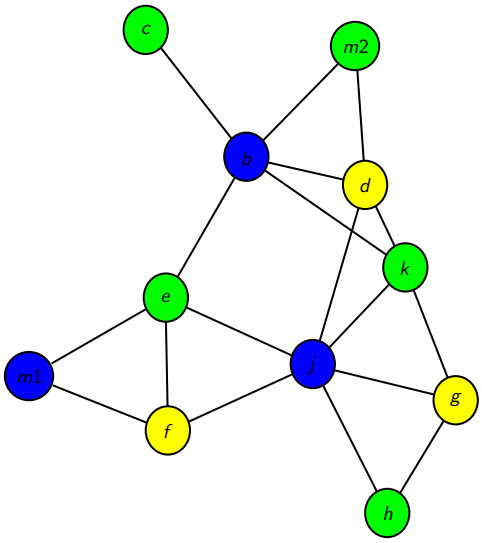


# Simplify, Spill, Select & Start Over

```
g = M[j+12]
h = k - 1
f = g * h
e = M[j+8]
m = M[j+16]
b = M[f]
c = e + 8
d = c
k = m + 4
j = b
```

$\Longrightarrow$

```
g = M[j+12]
h = k - 1
f = g * h
e = M[j+8]
m1 = M[j+16]
M[new] = m1
b = M[f]
c = e + 8
d = c
m2 = M[new]
k = m2 + 4
j = b
```



# Register Allocation in Practice

- Spills are uncommon on machines with large, uniform, register files.
- When a temporary is spilled, new instructions to load/store it are generated.
- The new instructions cut the temporaries lifetime into smaller parts, thus reducing interference.
- Register allocation is performed again on the modified code, potentially leading to other spills and restarts.
- **Precoloring:** Certain temporaries may have a predefined register (e.g. argument register). Then we color the interference graph, but start with preallocated colors for certain vertices.
- **Coalescing:** Temporaries that move data from each other can be constrained to have the same register, thereby avoiding data movement.

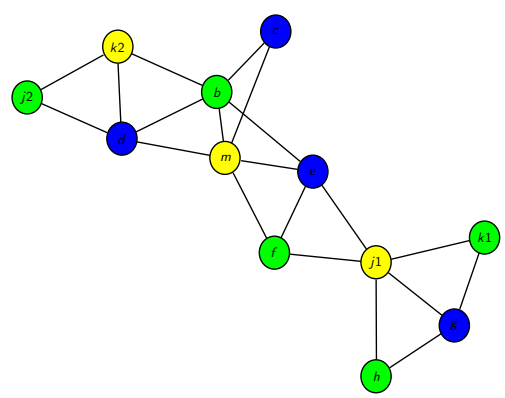




# SSA forms and Register Allocation

- SSA form eliminates spurious relationships in the interference graph.
- SSA may help avoid spills.

```
g = M[j+12]
h = k - 1
f = g * h
e = M[j+8]
m = M[j+16]
b = M[f]
c = e + 8
d = c
k = m + 4
j = b
```

$\Longrightarrow$

```
g = M[j1+12]
h = k1 - 1
f = g * h
e = M[j1+8]
m = M[j1+16]
b = M[f]
c = e + 8
d = c
k2 = m + 4
j2 = b
```