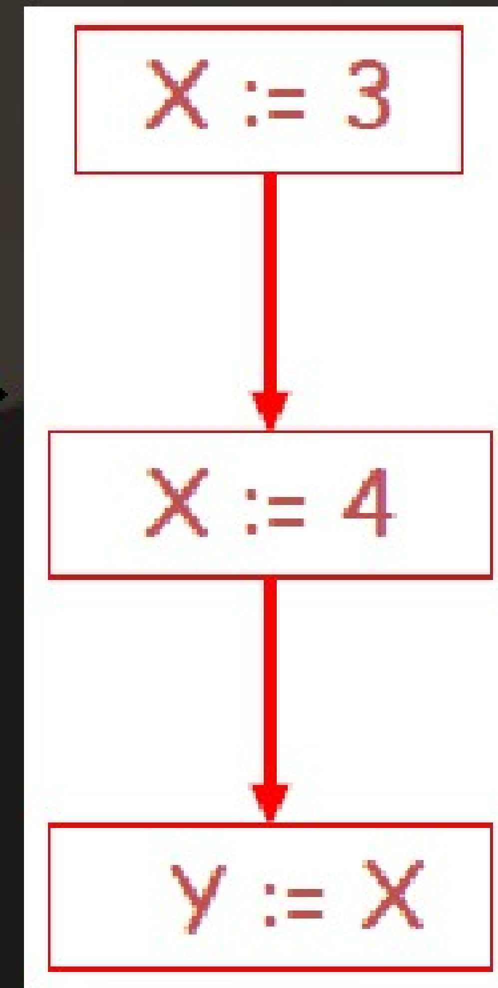
Lecture #33

Liveness Analysis & Register Allocation

Liveness Analysis



- The first value of x is *dead (never used)*
- The second value of x is *live (may be used)*

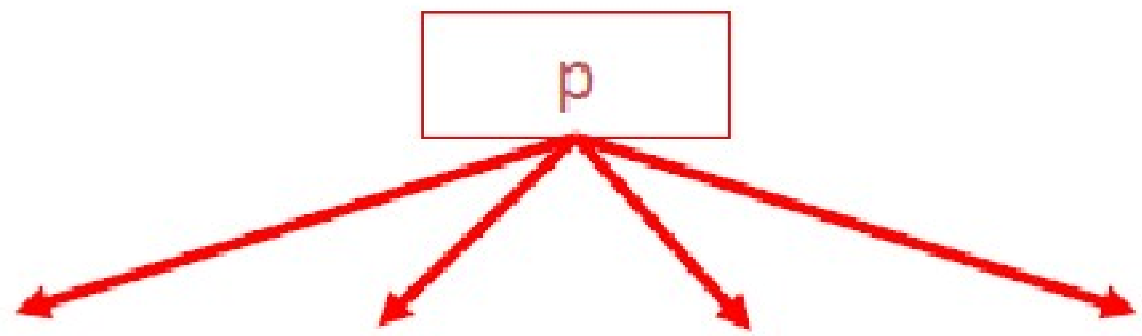
- A variable **x** is live at statement **s** if
  - There exists a statement **s'** that uses **x**
  - There is a path from s to **s'**
  - That path has no intervening assignment to **x**

- A statement **x = …** is dead code if **x** is dead after the assignment
  - Dead statements can be deleted from the program

Liveness Analysis

- Liveness can be expressed in terms of information transferred between adjacent statements, just as in copy propagation

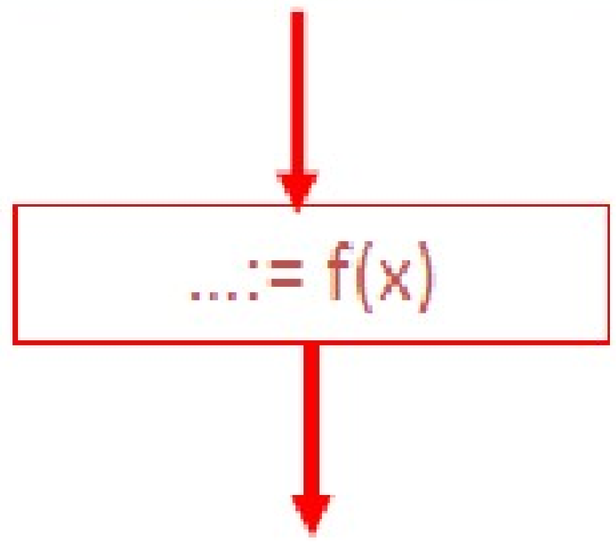
- Rule #1



$$L(p, x, out) = \vee \{ L(s, x, in) \mid s \text{ a successor of } p \}$$

Liveness Analysis

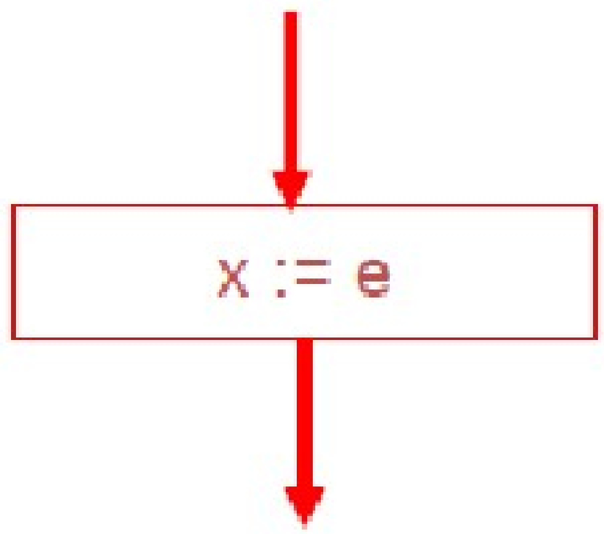
- Rule #2



L(s, x, in) = true if s refers to x on the rhs

Liveness Analysis

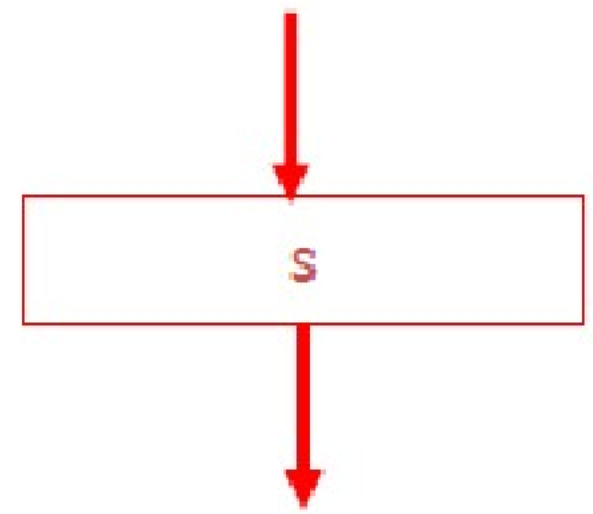
- Rule #3



L(x := e, x, in) = false if e does not refer to x

Liveness Analysis

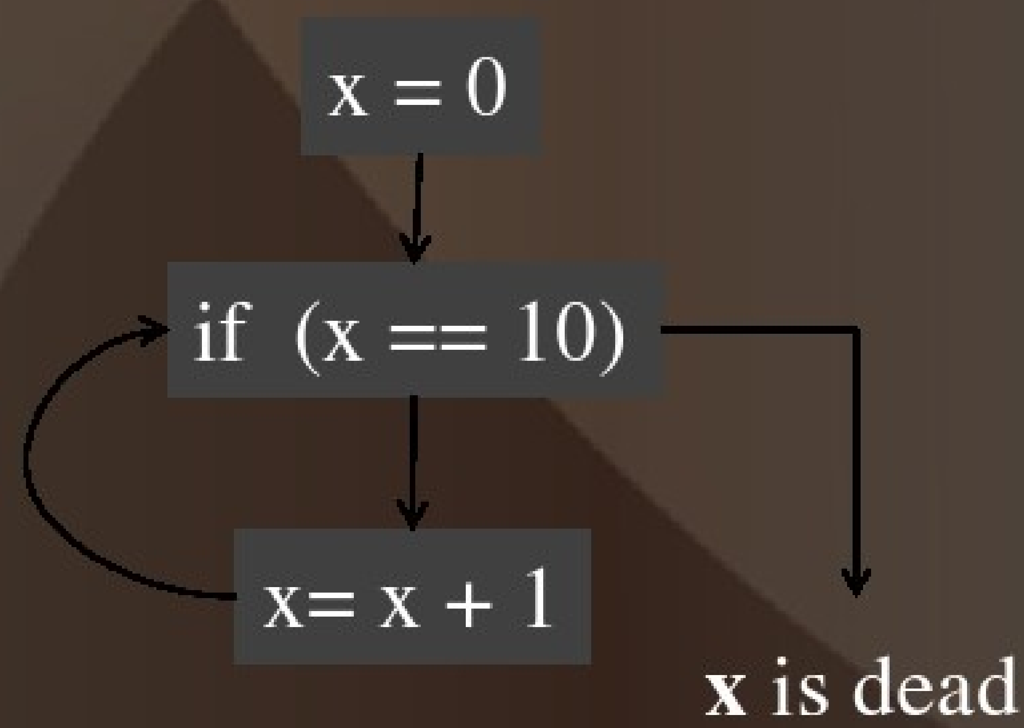
- Rule #4



L(s, x, in) = L(s, x, out) if s does not refer to x

Liveness Analysis

- Let all **L(…) = false** initially
- Repeat until all statements **s** satisfy rules 1-4
  - Pick **s** where one of 1-4 does not hold and update using the appropriate rule



Register Allocation

- The process of assigning a large number of target program variables onto a small number of CPU registers

- **Method:** Assign multiple temporaries to each register without changing the program behavior
  - Example:

a := c + d
e := a + b
f := e - 1

$\Rightarrow$

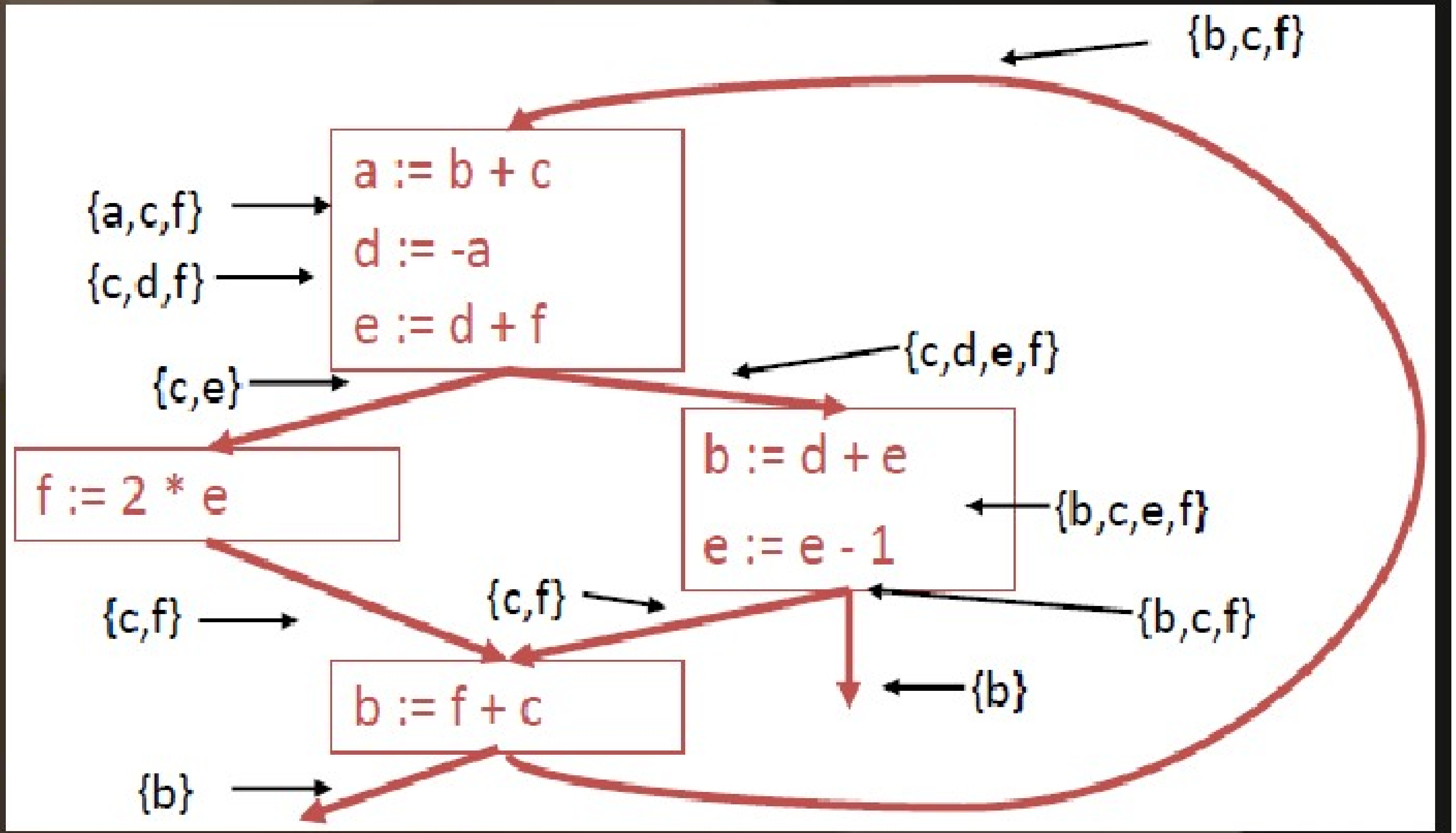
$r_1 := r_2 + r_3$
$r_1 := r_1 + r_4$
$r_1 := r_1 - 1$

a, **e** and **f** can all be allocated to the same register assuming **a** and **e** are dead after use

Register Allocation via Graph Colouring

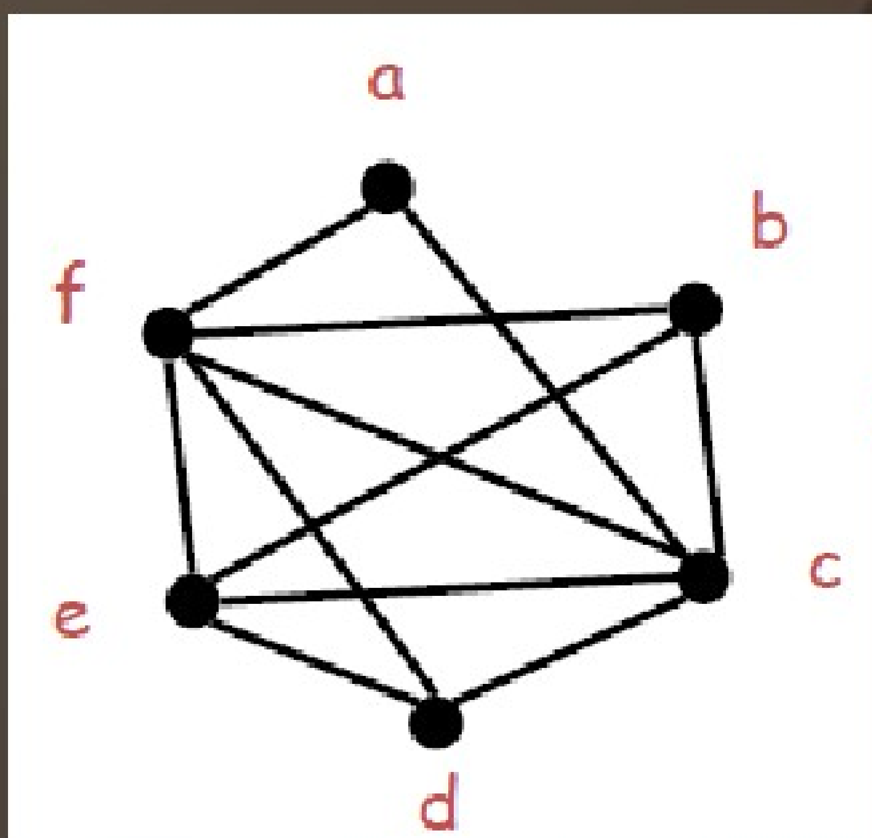
- **Basic Idea:** Two *temporary variables t_1 and t_2 can share the same register if at any point in the program at most one of t_1 or t_2 is live.*

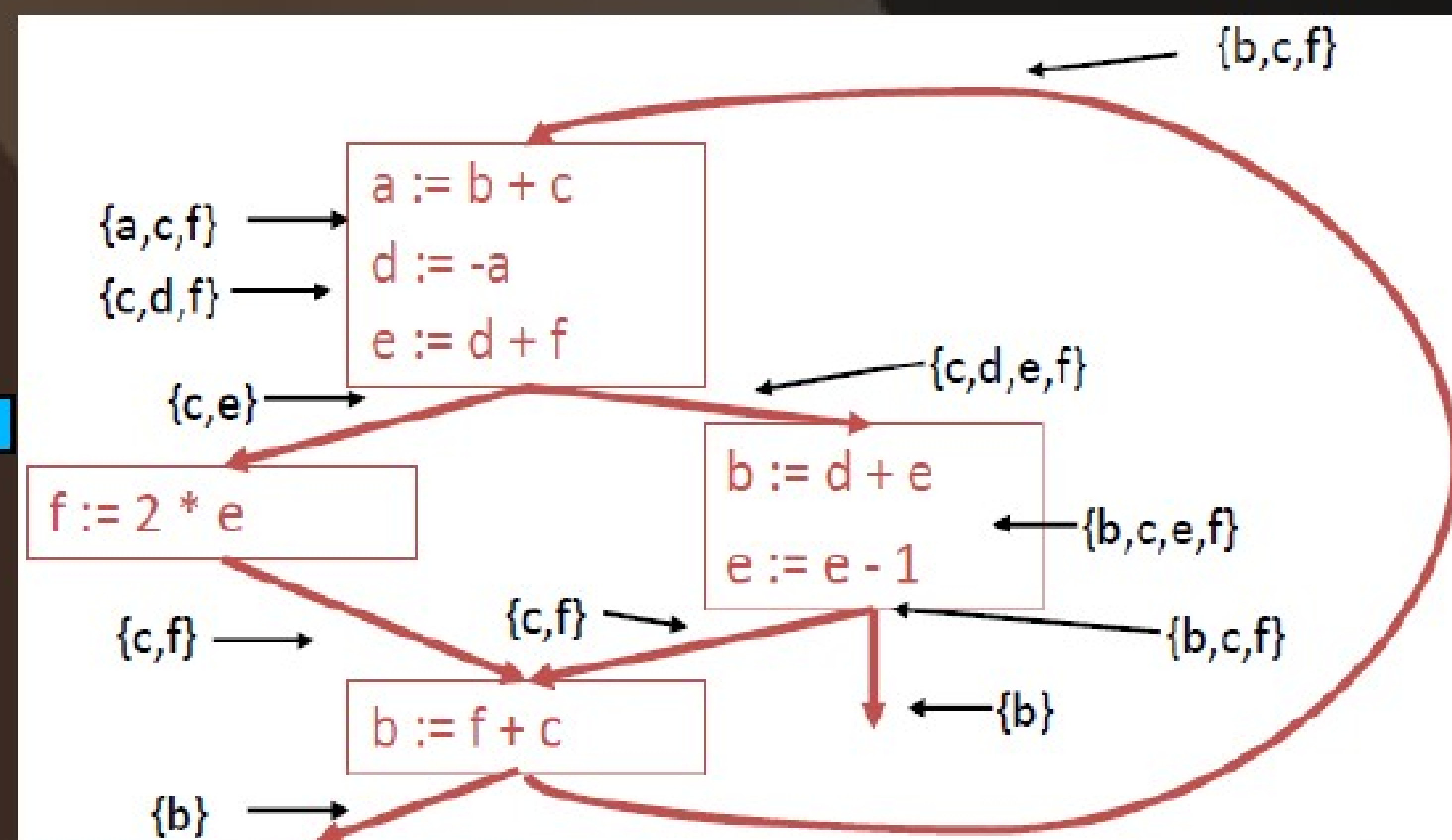
The variables alive simultaneously at each program point



Register Allocation via Graph Colouring

- **Register Interference Graph (RIG)**
  - An **undirected** graph
  - Each temporary variable is a **node**
  - Two temporary variables t_1 and t_2 share an **edge** if they are simultaneously alive at some program point





Register Allocation via Graph Colouring

- **Graph Colouring: A**n assignment of colours to nodes, such that nodes connected by an edge have different colors
- ***k*-coloring :** A coloring using at most *k* colours.
- **Chromatic number:** The smallest number of colours needed to colour a graph
- **Independent set:** A subset of vertices assigned to the same colour

- *k*-coloring is the same as a partition of the vertex set into *k* independent sets
    - The terms *k-partite* and *k-colourable* are equivalent

- *The graph colouring problem is **NP-Hard***
    - Heuristics needed to solve it

Register Allocation via Graph Colouring

- In the register allocation problem, colours = registers
- We need to assign colours (registers) to graph nodes (temporaries)
- Let k = number of machine registers
- If the RIG is *k-colourable* then there is a register assignment that uses no more than k registers

In our example RIG there is no coloring with less than 4 colours