

Low-Level Issues

Announcements

- Partners for project 2?

Last lecture

- Liveness analysis

Today

- Register allocation

Later

- More register allocation
- Instruction scheduling

Register Allocation

Problem

- Assign an unbounded number of **symbolic** registers to a fixed number of **architectural** registers
- Simultaneously live data must be assigned to different architectural registers

Goal

- Minimize overhead of accessing data
 - Memory operations (loads & stores)
 - Register moves

Scope of Register Allocation

Expression

Local

Loop



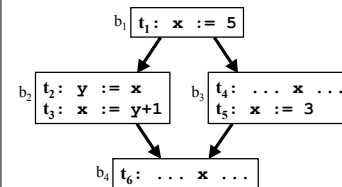
Global

Interprocedural

Granularity of Allocation

What is allocated to registers?

- Variables
- Live ranges/Webs (*i.e.*, du-chains with common uses)
- Values (*i.e.*, definitions; same as variables with SSA)



Variables: 2 (x & y)

Live Ranges/Webs: 3 ($t_1 \rightarrow t_2, t_4$;

$t_2 \rightarrow t_3$;

$t_3, t_5 \rightarrow t_6$)

Values: 4 ($t_1, t_2, t_3, t_5, \phi(t_3, t_5)$)

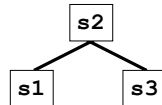
What are the tradeoffs?

Each allocation unit is given a symbolic register name (*e.g.*, s_1, s_2 , etc.)

Global Register Allocation by Graph Coloring

Idea [Cocke 71], First allocator [Chaitin 81]

1. Construct **interference graph** $G=(N,E)$
 - Represents notion of "simultaneously live"
 - Nodes are units of allocation (e.g., variables, live ranges)
 - \exists edge $(n_1, n_2) \in E$ if n_1 and n_2 are simultaneously live
 - Symmetric (not reflexive nor transitive)
2. Find **k -coloring** of G (for k registers)
 - Adjacent nodes can't have same color
3. **Allocate** the same register to all allocation units of the same color
 - Adjacent nodes must be allocated to distinct registers

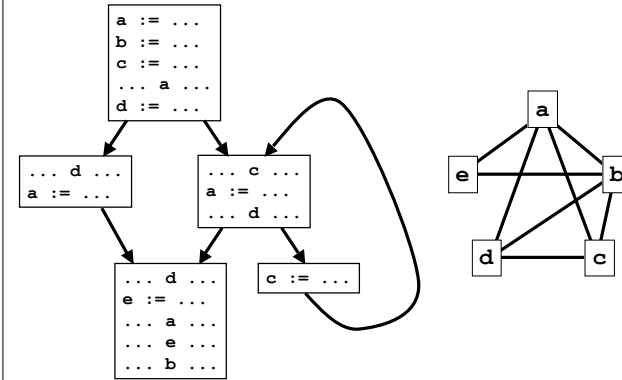


CS553 Lecture

Register Allocation I

6

Interference Graph Example (Variables)



CS553 Lecture

Register Allocation I

7

Computing the Interference Graph

Use results of live variable analysis

```

for each symbolic-register  $s_i$  do
  for each symbolic-register  $s_j$  ( $j < i$ ) do
    for each def  $\in$  {definitions of  $s_i$ } do
      if ( $s_j$  is live at def) then
         $E \leftarrow E \cup (s_i, s_j)$ 
    
```

Options

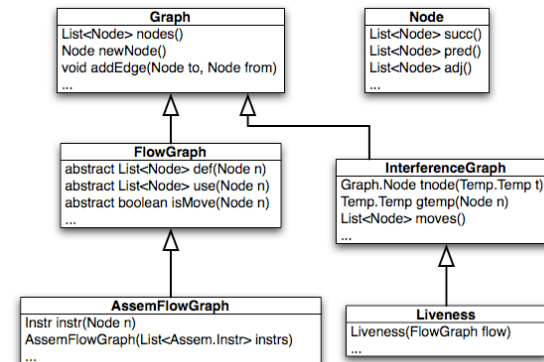
- pg. 217 in Tiger book, treat all instructions the same
- pg. 213-214 in Tiger book, treat MOVE instructions special
- which is better?

CS553 Lecture

Register Allocation I

8

Liveness in the MiniJava compiler



CS553 Lecture

Register Allocation I

9

Allocating Registers Using the Interference Graph

K-coloring

- Color graph nodes using up to k colors
- Adjacent nodes must have different colors

Allocating to k registers = finding a k -coloring of the interference graph

- Adjacent nodes must be allocated to distinct registers

But...

- Optimal graph coloring is NP-complete
 - Register allocation is NP-complete, too (must approximate)
- What if we can't k -color a graph? (must **spill**)

Register Allocation: Spilling

If we can't find a k -coloring of the interference graph

- Spill variables (nodes) until the graph is colorable




Choosing variables to spill

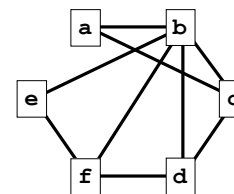
- Choose arbitrarily or
- Choose least frequently accessed variables
- Break ties by choosing nodes with the most conflicts in the interference graph
- Yes, these are heuristics!

Simple Greedy Algorithm for Register Allocation

```
for each  $n \in N$  do           { select  $n$  in decreasing order of weight }
  if  $n$  can be colored then
    do it                     { reserve a register for  $n$  }
  else
    Remove  $n$  (and its edges) from graph { allocate  $n$  to stack (spill) }
```

Example



Attempt to 3-color this graph ( ,  , )

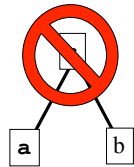


Arbitrary order:
a
b
c
d
f
e

What if you use a different order?

Example

Attempt to 2-color this graph ( , )



Weighted order:
a
b
c

Improvement #1: Simplification Phase [Chaitin 81]

Idea

- Nodes with $< k$ neighbors are guaranteed colorable

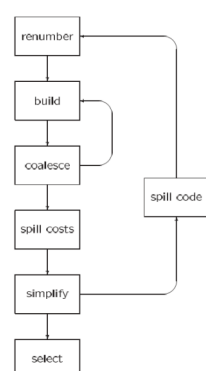
Remove them from the graph first

- Reduces the degree of the remaining nodes

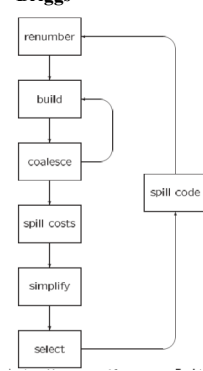
Must spill only when all remaining nodes have degree $\geq k$

Simplifying Graph Allocators

Chaitin



Briggs



Algorithm [Chaitin81]

```
while interference graph not empty do
  while  $\exists$  a node  $n$  with  $< k$  neighbors do
    Remove  $n$  from the graph
    Push  $n$  on a stack
  if any nodes remain in the graph then { blocked with  $\geq k$  edges }
    Pick a node  $n$  to spill { lowest spill-cost or }
    Add  $n$  to spill set { highest degree }
    Remove  $n$  from the graph
  if spill set not empty then
    Insert spill code for all spilled nodes { store after def; load before use }
    Reconstruct interference graph & start over
  while stack not empty do
    Pop node  $n$  from stack
    Allocate  $n$  to a register
```

More on Spilling

Chaitin's algorithm restarts the whole process on spill

- Necessary, because spill code (loads/stores) uses registers
- Okay, because it usually only happens a couple times

Alternative

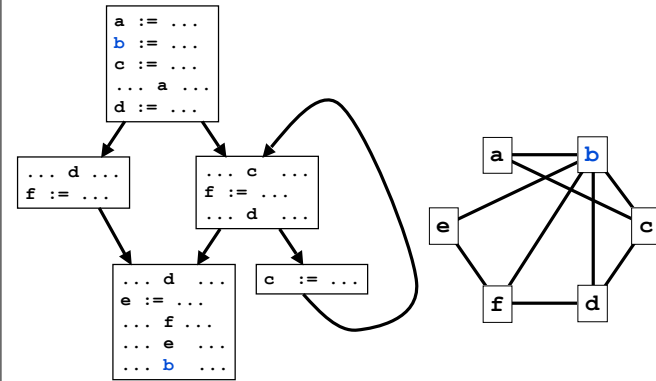
- Reserve 2-3 registers for spilling
- Don't need to start over
- But have fewer registers to work with

CS553 Lecture

Register Allocation I

18

Spilling (Original CFG and Interference Graph)

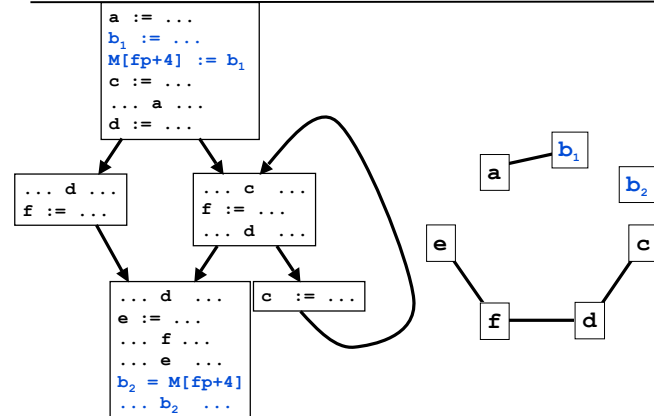


CS553 Lecture

Register Allocation I

19

Spilling (After spilling b)



CS553 Lecture

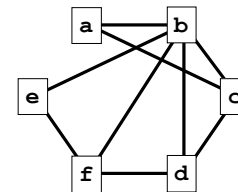
Register Allocation I

20

Example

Attempt to 3-color this graph (■ , ■ , ■)

Stack:
d
c
b
f
a
e



Possible order:
e
a
f
b
c
d

CS553 Lecture

Register Allocation I

21

Next Time

Lecture

- More register allocation
 - Allocation across procedure calls