Register Allocation

Registers: small number of fast memories, close to processor

Register to register operations fast

Register allocation: ***Determine which values should be kept in registers at each program point***

Usually performed on LIL

Addressing, loads and stores, temps exposed

Candidates for register allocation

Large constants, local and non-local vars, array elements,...

What shouldn't be allocated in registers?

Register Assignment

Which register should value be placed in?

May be classes of registers

Integer, floating point,...

Some operations may not use registers

Ex. I/O, communication,...

Some registers may have special uses

Ex. procedure calls, stack pointer, string manip.,...

What if can't fit in the number of registers available?

Local Register Allocation

Use registers to hold values during basic block
Load values at beginning of basic block
Store (live) values at end of basic block

What if there are not enough registers?

Sav(V,B) , estimate of cost savings of var B in reg during BB B

Sav(V,B) = # Uses(V) * Usesave + # Defs (V)* Defsave
- Loadcost - Storecost

How determine Usesave, Defsave, Loadcost, Storecost?

Put variables with highest savings in registers

What about Loops?

Basic blocks in inner loops before outer

Take loop nesting, execution frequency into account

Use profiling info

Make static estimate

$$Sav(V,L) = 10^{d} * \sum_{B \text{ in } L} Sav(V,B)$$

where d is the depth of nesting of L

Approach used in IBM 360/370 compilers

Can also try to reuse same register as predecessor(s)

Global Register Alloc. (Graph Coloring)

Cocke et al, IBM

Overview

1. ***Allocate symbolic registers to assignable objects***
2. ***Determine the candidates for allocation***
3. ***Construct the Interference Graph***

   Nodes: allocatable objects

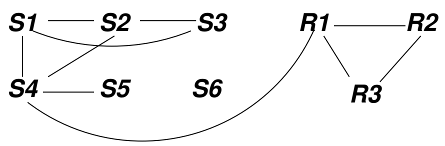
   Undirected edges: between nodes that cannot be assigned same reg.
4. ***Color the IG with R colors, where R = # available regs, so nodes connected by an edge have different colors***
5. ***Allocate objects with the same color to the same real reg.***

Global Register Alloc. Example

x:=2	S1:=2
y:=4	S2:=4
w:=x+y	S3:=S1+S2
z:=x+1	S4:=S1+1
u:=x*y	S5:=S1*S2
x:=z*2	S6:=S4*2

y, w dead

3 Real Registers, R1, R2, R3 ; z cannot occupy R1

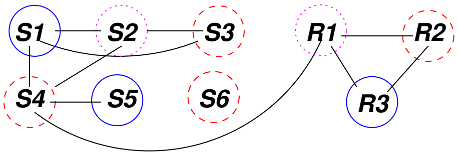


Global Register Alloc. Example

x:=2	S1:=2	R1:=2
y:=4	S2:=4	R2:=4
w:=x+y	S3:=S1+S2	R3:=R1+R2
z:=x+1	S4:=S1+1	R3:=R1+1
u:=x*y	S5:=S1*S2	R1:=R1*R2
x:=z*2	S6:=S4*2	R2:=R3*2

y, w dead

3 Real Registers, R1, R2, R3 ; z cannot occupy R1



Assignable Objects

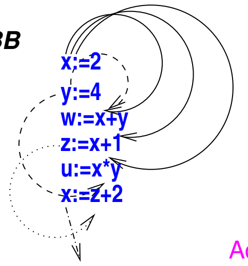
Candidates:

Variable names

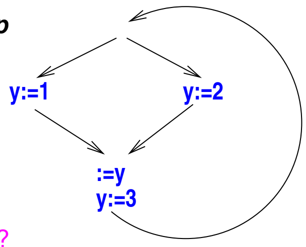
Def-use edge

Web: maximal union of def-use chains that share common use

Ex. ***BB***



Loop



Advantages?

Interference Graph

Nodes: ***Webs, machine registers***

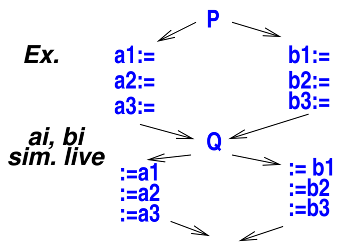
Undirected Edges***:***

X ——— ***Y***

X and Y simultaneously live , or

Y live at a def of X

Which is better?



Interference Graph Representation

Can use large amount space, access time costly

What operations?

Test if 2 nodes adjacent

Find all nodes adjacent to a given node

Find # nodes adjacent to a given node

Adjacency Matrix

(i,j) entry = true iff Si, Sj adjacent in IG

Lower triangular matrix

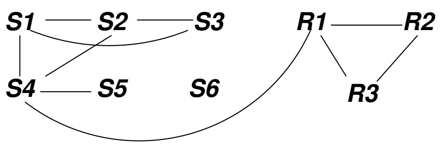
Adjacency Lists

Adjacency Matrix Example

x:=2
y:=4
w:=x+y
z:=x+1
u:=x*y
x:=z*2

S1:=2
S2:=4
S3:=S1+S2
S4:=S1+1
S5:=S1*S2
S6:=S4*2

	R1	R2	R3	S1	S2	S3	S4	S5
R2	t							
R3	t	t						
S1	f	f	f	f				
S2	f	f	f	t	f			
S3	f	f	f	t	t	f		
S4	t	f	f	t	t	f	f	
S5	f	f	f	f	f	f	t	f
S6	f	f	f	f	f	f	f	f



Register Coalescing

Copy propagation on register copy instructions Sj:=Si, where Sj, Si do not interfere

Modify Instructions:

Write to Si	→	*Write to Sj*
Sj := Si	→	
use of Si	→	*use of Sj*

Modify IG:

Combine Si, Sj nodes into single node

Remove any interference edges with live Sk at copy Sj := Si

Spilling Registers

May not be possible to put all desirable objects in registers

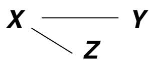
If cannot, must spill: store value, then later reload from memory

Spilling may decrease # registers needed

Split one node into 2

Ex.

def X
def Y
use Y
def Z
use Z
use X



def X
store X
def Y
use Y
def Z
use Z
load X
use X

R-Coloring the ISG

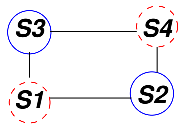
NP-complete problem for R>2

Heuristics needed

Heuristic 1: Remove nodes with degree < R from graph.
Resulting graph is R-colorable iff original was.

Note: graph may be R-colorable, but all nodes have degree R.

Ex



2-colorable

Heuristic 1 may by itslef make graph R-colorable, for real R

Put nodes on stack as remove; if graph empty, then R-colorable

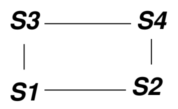
R-Coloring the ISG

Heuristic 2: If no nodes left of degree < R, remove node with
$$\min \frac{\text{spill cost}\ (N)}{\text{current deg}(N)}$$

Put nodes on stack as remove.

When done, color as pop off the stack

Ex



S1 S2 S3 S4

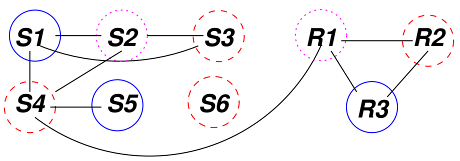
In this case, can color with 2 colors

Global Register Alloc. Example

```
x:=2
y:=4
w:=x+y
z:=x+1
u:=x*y
x:=z*2
```

Stack

R1 R2 R3 S4 S2 S1 S3 S5 S6



Summary, Global Register Allocation

1. Construct webs
2. Build Adjacency matrix
3. Coalesce registers
4. Perform Heuristic #1 as much as possible. If IG empty, go to 6.
5. Perform Heuristic #2, if needed.
6. Color the nodes in reverse stack order. If R-colorable, gen. code.

   If not, then generate spill code for top of stack, and go to 1.