

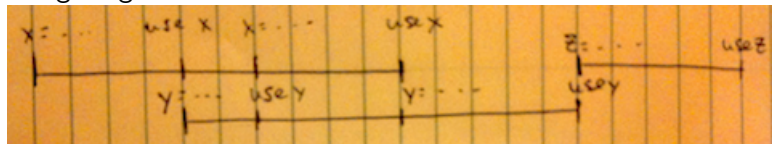
# Modern architectures make register allocation simpler

Paul Khuong (pvk@pvk.ca)

December 14, 2009

# Classic register allocation (colouring)

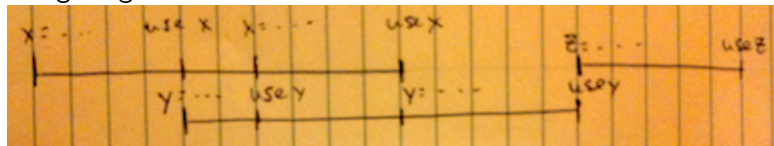
Assign registers to variables for the duration of their lifetimes



Assignment:

# Classic register allocation (colouring)

Assign registers to variables for the duration of their lifetimes

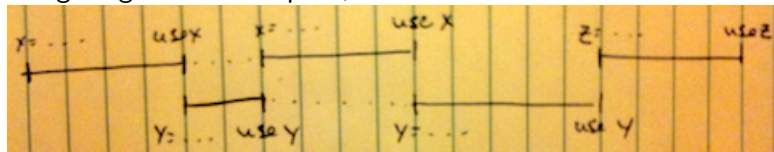


Assignment:

- ▶ x: r1
- ▶ y: r2
- ▶ z: r1

# Colour values, not variables

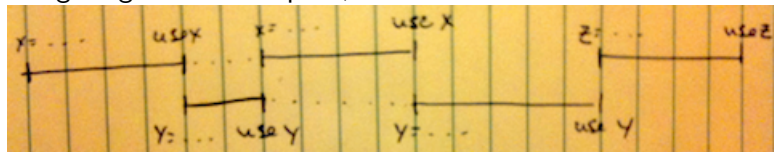
Assign registers to lifespans, not variables



Assignment:

# Colour values, not variables

Assign registers to lifespans, not variables

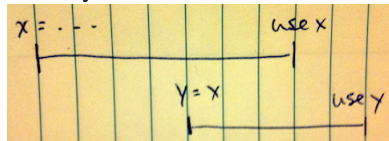


Assignment: everything fits in r1!

# What about copies?

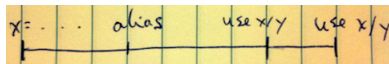
Insight: variables are but names for values!

Naïvely



Needs 2 registers

“Virtual” copy



Everything fits in 1 register

## $\phi$ -functions

Permute or copy values to specific registers/stack locations to meet the successor's expectations.

Predecessor	Successor
$x, y : r_1$	$r_1 : x$
$z : r_2$	$r_2 : y$
	$r_3 : z$

Implementation

$$r_2 \rightarrow r_3$$
$$r_1 \rightarrow r_2$$

(Simple case :))

# Live-range splitting/coalescing

**Split** gives more freedom to the allocator, by moving/spilling values around code sequences where they are useless

**Coalesce** improves the generated code by minimising the number of moves (ideally, turns all  $\phi$ -functions into identities)

# Outline

Overview

New tradeoffs

Solution

Temporary-free  $\phi$ -functions

No-opifying  $\phi$ -functions

Conclusion

# Before

- ▶ Bottleneck: executing instructions
- ▶ Goal: minimise number of instructions, including moves

# Now

- ▶ Bottleneck: operating on memory (moves are nearly free with a superscalar and OOO  $\mu$ arch)
- ▶ Goal: minimise number of spills

Small print: I\$ and decoder bandwidth can be an issue.

# Strategy

Ensure that  $\#$  registers is equal to  $\#$  live values.

SSA already ensures that we only consider live values.

As a side-effect, this makes it easier to take regalloc into account during codegen. It could also be extended to handle x87 natively (FWIW).

# Execution

Insert  $\phi$ -functions after each VOP (takes care of wired arguments, e.g. MUL).

Implement  $\phi$ -functions without temporaries when possible.

Make sure as many  $\phi$ -functions as possible are identities (heuristic).

## Simple $\phi$ -functions

- ▶ Some values are useless (or some register is unused): easy, we can use temporaries
- ▶ Some values are copied: easy, some values are useless
- ▶ Values on the stack: mostly easy, stack frames can be expanded

## Interesting $\phi$ -functions

Everything in registers, and we have a perfect permutation (no useless nor copied value).

Borrow a trick from algebra:

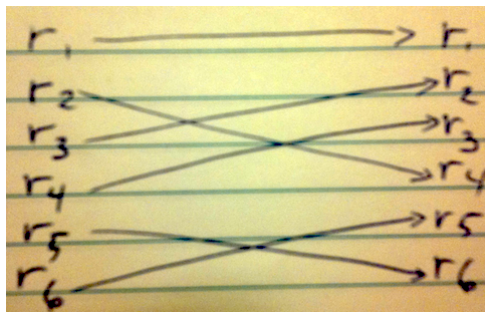
## Interesting $\phi$ -functions

Everything in registers, and we have a perfect permutation (no useless nor copied value).

Borrow a trick from algebra: the cyclic representation shows us how to decompose the permutation into a set of rotations, which we can implement as series of swaps.

What if we don't have XCHG? SSE registers still have XOR, so we can use the old trick.

## Example



Cycle form:  $(r_1) \circ (r_2 r_4 r_3) \circ (r_5 r_6)$

Implementation:

$r_5 \leftrightarrow r_6$

$r_2 \leftrightarrow r_4$

$r_2 \leftrightarrow r_3$

Worst-case:  $\#REG - 1$  swaps (2 fewer than moves with temporaries!)

# Turning $\phi$ -functions into no-ops is Hard

That's equivalent to colouring! Heuristic:

- ▶ Propagate forward across “key” CFG arcs (if key arcs form a tree, there is no join node)
- ▶ Propagate wired arguments backward (locally)

# Take-home message

- ▶ Regalloc isn't complicated
- ▶ Heuristics are needed at some point for a practical compiler
- ▶ Have to identify when swaps are preferable to hitting memory (can be surprising with store-to-load forwarding), or to using temporaries (renaming could work less well with XCHG or XOR)
- ▶ If I find time to do this in the next 18 months, it's because I'm being irresponsible. . .