Lec 3 Register Allocation

IR tree —(instruction selection)→ abs. asm (reg allocations) ← we're here → x86 asm

The problem

$d \leftarrow s_1 * s_2$ ⇝ `r11d ← exd * 4(rsp)` (stack location)

Register interference

```
x ← 14
y ← 15 + x
z ← x + y
return z
```

can't use same reg for x, y

```
x ← 14
y ← 15 + x
z ← 4 + y
return z
```

fine to reuse reg for x and y

Intermediate repr for reg allocation

▷
- 3-addr abs. asm
- 2-addr abs. asm
- abs. x86

3-addr	2-addr	abs. x86
$d \leftarrow s_1 + s_2$	$d \leftarrow s_1$ $d \leftarrow d + s_2$	`movl s1, d` `addl s2, d`

$d \leftarrow s_1 / s_2$	$d \leftarrow s_1$
	$d \leftarrow d / s_2$
		`movl s1, eax`
		`cdtt`
		`idivl s2`
		`movl eax, d`

`idivl s2`: edx : eax / s2 → put in eax

▷ reserve a reg for later conversion

$d \leftarrow s_1 + s_2$ ⟶

```
movl s1, r11d
addl s2, r11d
movl r11d, d
```

← Note in general some of them may be on stack. If all registers, more efficient to:

```
movl s1, d
addl s2, d
```

whereas if we had 2-addr forms

$d \leftarrow s_1$

$d \leftarrow d + s_2$

then both s_1, d could be spilled

Spill minimisation

later: possible to do lifespan splitting to reduce interference

Today: graph-based, greedy allocator

Steps
1. Build interference graph
   - V = {regs} ∪ {temps}
   - E = {interferences} ← note the regs already form a clique

Ex. $t_1 - t_2 - t_3 - eax$

2. Find k-colouring with minimised k

Ex. $t_1 - t_2 - t_3 - eax$

3. Assign colours to regs & stack location
   - if n colour ≤ n regs then good
   - else decide which colour to spill

Implementation

Problem... deciding if two temps interfere is UNDECIDABLE!

Proof HALT ≤ INTERFERE

```
M_HALT (⟨M, x⟩):
  M'(y):
    x ← 5   ] distinct from temps
    y ← 6   ] inside M's code
    run M(x)
    ret ← x + y
    return
  return M_INTERFERE (⟨M', ε⟩)
```

Workaround: over approximate — be safe when unsure

▷ Liveness Analysis

in := right before line
at/on := on the line ? ← confusing
out := right after line

Def temp t is live at line l if t might be used in future computation

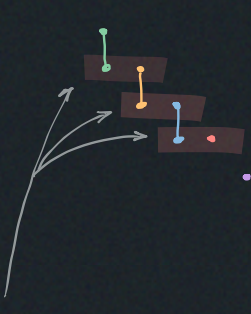
▷ Work backward strategy for L1

temp t is live at line l if any of:
- t is read at l
- t is live at line l+1 and not written at line l

Ex.

```
x1 ← 1
x2 ← 2
x3 ← x1 + x2
x4 ← x3 + x2
x5 ← x3 + x4
ret ← x5
return
```

live-in
x_1
x_2, x_1
x_3, x_2
x_3, x_4
x_5
ret



Turning this into graph (options)
1. edge for overlapping live ranges at any point viz. both alive on same line — overapproximates
2. edge if any of:
   - For every instruction $d \leftarrow s_1 \oplus s_2$, if t is live-in of next instruction, add edge {d, t}
     - hope is that s_1, s_2 won't be live again
   - For every move $d \leftarrow s$, add edge {d, t} for $t \notin \{s, d\}$ that is live-in of next instruction
     - if d, s not changed in future, d = s so one can give them same reg
     - but we don't want to write to t's reg, if t needs to be alive

Either options gives valid graph.
Note less edges ⇏ better graph