Register Allocation, iii

Liveness analysis & Building the interference graph

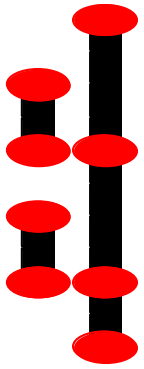
Liveness analysis

Find **live ranges** for each variable and register:

- Starts when var is assigned
- Ends at last reference
- Two assignments ⇒ two live ranges

Liveness analysis

```
                x  y
(:f
 0 0
 (y <- 1)
 (x <- 2)
 (y += x)
 (x <- 3)
 (y += x)
 (rax <- y)
 (return))
```



The two assignments to **x** count as two separate live ranges (and thus can live in different registers, if necessary)

Liveness analysis

Compute live ranges via fixed point:

- Define which variables are set and which are used, for each instruction: **kill** & **gen** functions
- Specify how instructions transmit live values around the program: **in** & **out** functions
- Iterate until nothing changes

kill, gen : i → register set

kill(s) = {all assigned registers in stm s}

gen(s) = {all referenced registers in stm s}

Naive (wrong) gen & kill

		gen	kill
1:	`(x2 <- rdi)`	(rdi)	(x2)
2:	`(x2 *= x2)`	(x2)	(x2)
3:	`(dx2 <- x2)`	(x2)	(dx2)
4:	`(dx2 *= 2)`	(dx2)	(dx2)
5:	`(tx <- rdi)`	(rdi)	(tx)
6:	`(tx *= 3)`	(tx)	(tx)
7:	`(rax <- dx2)`	(dx2)	(rax)
8:	`(rax += tx)`	(rax tx)	(rax)
9:	`(rax += 4)`	(rax)	(rax)
10:	`(return)`	(rax)	()

Wrong because the calling convention counts as references and assignments to registers; when we add in the right mix of assignments and references implicitly to `return`, `call`, and `tail-call` instructions, we automatically enforce the calling convention

Calling convention gen/kill overview

caller save `r10 r11 r8 r9 rax rcx rdi rdx rsi`
args `rdi rsi rdx rcx r8 r9`
callee save `r12 r13 r14 r15 rbp rbx`
result `rax`

	gen	kill
`(call u nat)`	`u` args	caller save result
`(tail-call u nat)`	`u` args callee save	
`(return)`	result callee save	

Call may kill caller save regs

	gen	kill
`(call u nat)`	`u` args	caller save result
`(tail-call u nat)`	`u` args callee save	
`(return)`	result callee save	

```
(a <- 2)
(call :f 0)
(a *= a)
```

Since the caller is responsible for saving the caller save registers, e.g. **`rcx`**, then **`:f`** is free to clobber it. Thus we must be sure not to use **`rcx`** for **`a`**.

Making the **`call`** kill **`rcx`** means that a new live range starts at the call instruction and thus **`a`** and **`rcx`** have overlapping live ranges and thus won't share a register

Return gens result

	gen	kill
`(call u nat)`	`u` args	caller save result
`(tail-call u nat)`	`u` args callee save	
`(return)`	result callee save	

```
(rax <- 1)
(a <- rdi)
(a *= a)
((mem b 8) <- a)
(return)
```

Must make sure that **a** is not allocated into the result register **rax**. With a reference (aka gen) at the **return** then the live range spans the assignment to **a** and thus **a** won't go into **rax**

tail-call/return gen callee save

	gen	kill
`(call u nat)`	`u` args	caller save result
`(tail-call u nat)`	`u` args callee save	
`(return)`	result callee save	

```
(rax <- 1)
(a <- rdx)
(a *= a)
((mem b 4) <- a)
(return)
```

Enforces the calling convention; specifically that the callee save registers are actually saved. In the code on the left, `a` must not be in `rsi`

With a reference and no kill, the live range will span the entire function and thus ensure that `rsi` conflicts with all user variables

Call kills return

	gen	kill
`(call u nat)`	`u` args	caller save result
`(tail-call u nat)`	`u` args callee save	
`(return)`	result callee save	

```
(a <- 1)
(call :f)
(b <- a)
```

If we put `a` into `rax` then the call to `:f` would clobber it to save the return value, so we must avoid that by treating the `call` as a kill to the return register, `rax`

(Yep, this is the second reason to do this; other calling conventions may not have both reasons)

Call/tail-call gen arguments

	gen	kill
`(call u nat)`	`u` args	caller save result
`(tail-call u nat)`	`u` args callee save	
`(return)`	result callee save	

```
(rsi <- 1)
(a <- 2)
(call :f 1)
```

If we put `a` into `rsi` then the call to `:f` would get the wrong arguments. Avoid this by making `call` and `tail-call` refer to (i.e., gen) the argument registers that they use (based on the arity).

Probably your compiler won't generate code like this, but spilling could put you into this kind of situation.

Runtime system calls

	gen	kill
`(call u nat)`	`u` args	caller save result
`(tail-call u nat)`	`u` args callee save	
`(return)`	result callee save	

```
(a <- 1)
(rdi <- 1)
(call print 1)
(b += a)
```

The chart shows only `(call u nat)` instructions, but the same rules apply for calls to the runtime system (`read`, `print`, `allocate`, and `array-error`)

in & out

in, out : f Nat → register set

in(f,n) = { all registers live right before f[n] runs }

out(f,n) = { all registers live right after f[n] runs }

in and **out** map the number of an instruction in some given function, f, to a set of registers; the notation f[n] means the n^{th} instruction of f

This is a specification of **in** and **out**, not how to compute them.

in & out

in, out : f Nat → register set

in(f,n) = **gen**(f[n]) ∪ (**out**(f[n]) **- kill**(f[n]))

out(f,n) = ∪{**in**(f,m) | m ∈ **succ**(f,n)}

succ : f Nat → Nat set
succ(f,n) = { index of each instruction that can follow n in f }

This is an equivalent specification of **in** and **out** but this time in terms of **gen**, **kill**,and **succ**

in & out

in, out : f Nat → register set

in(f,n) = **gen**(f[n]) ∪ (**out**(f[n]) **- kill**(f[n]))

out(f,n) = ∪{**in**(f,m) | m ∈ **succ**(f,n)}

succ : f Nat → Nat set
succ(f,n) = { index of each instruction that can follow n in f }

Compute **in** and **out** by iterating the equations until we reach a fixed point. Starting out with the assumption that they map everything to the empty set. Then repeatedly apply the right hand sides until nothing changes.

in & out

in, out : f Nat → register set

in(f,n) = **gen**(f[n]) ∪ (**out**(f[n]) **- kill**(f[n]))

out(f,n) = ∪{**in**(f,m) | m ∈ **succ**(f,n)}

succ : f Nat → Nat set
succ(f,n) = { index of each instruction that can follow n in f }

That result will satisfy the definitions of **in** and **out** (and it will it be the smallest solution)

Liveness, straight-line code

		in	out
1:	`(x2 <- rdi)`	()	()
2:	`(x2 *= x2)`	()	()
3:	`(dx2 <- x2)`	()	()
4:	`(dx2 *= 2)`	()	()
5:	`(tx <- rdi)`	()	()
6:	`(tx *= 3)`	()	()
7:	`(rax <- dx2)`	()	()
8:	`(rax += tx)`	()	()
9:	`(rax += 4)`	()	()
10:	`(return)`	()	()

Liveness, straight-line code

		in	out
1:	`(x2 <- rdi)`	(rdi)	()
2:	`(x2 *= x2)`	(x2)	()
3:	`(dx2 <- x2)`	(x2)	()
4:	`(dx2 *= 2)`	(dx2)	()
5:	`(tx <- rdi)`	(rdi)	()
6:	`(tx *= 3)`	(tx)	()
7:	`(rax <- dx2)`	(dx2)	()
8:	`(rax += tx)`	(rax tx)	()
9:	`(rax += 4)`	(rax)	()
10:	`(return)`	(r12 r13 r14 r15 rax rbp rbx)	()

Liveness, straight-line code

		in	out
1:	`(x2 <- rdi)`	(rdi)	(x2)
2:	`(x2 *= x2)`	(x2)	(x2)
3:	`(dx2 <- x2)`	(x2)	(dx2)
4:	`(dx2 *= 2)`	(dx2)	(rdi)
5:	`(tx <- rdi)`	(rdi)	(tx)
6:	`(tx *= 3)`	(tx)	(dx2)
7:	`(rax <- dx2)`	(dx2)	(rax tx)
8:	`(rax += tx)`	(rax tx)	(rax)
9:	`(rax += 4)`	(rax)	(r12 r13 r14 r15 rax rbp rbx)
10:	`(return)`	(r12 r13 r14 r15 rax rbp rbx)	()

Liveness, straight-line code

		in	out
1:	`(x2 <- rdi)`	(rdi)	(x2)
2:	`(x2 *= x2)`	(x2)	(x2)
3:	`(dx2 <- x2)`	(x2)	(dx2)
4:	`(dx2 *= 2)`	(dx2 rdi)	(rdi)
5:	`(tx <- rdi)`	(rdi)	(tx)
6:	`(tx *= 3)`	(dx2 tx)	(dx2)
7:	`(rax <- dx2)`	(dx2 tx)	(rax tx)
8:	`(rax += tx)`	(rax tx)	(rax)
9:	`(rax += 4)`	(r12 r13 r14 r15 rax rbp rbx)	(r12 r13 r14 r15 rax rbp rbx)
10:	`(return)`	(r12 r13 r14 r15 rax rbp rbx)	()

Liveness, straight-line code

		in	out
1:	`(x2 <- rdi)`	(rdi)	(x2)
2:	`(x2 *= x2)`	(x2)	(x2)
3:	`(dx2 <- x2)`	(x2)	(dx2 rdi)
4:	`(dx2 *= 2)`	(dx2 rdi)	(rdi)
5:	`(tx <- rdi)`	(rdi)	(dx2 tx)
6:	`(tx *= 3)`	(dx2 tx)	(dx2 tx)
7:	`(rax <- dx2)`	(dx2 tx)	(rax tx)
8:	`(rax += tx)`	(rax tx)	(r12 r13 r14 r15 rax rbp rbx)
9:	`(rax += 4)`	(r12 r13 r14 r15 rax rbp rbx)	(r12 r13 r14 r15 rax rbp rbx)
10:	`(return)`	(r12 r13 r14 r15 rax rbp rbx)	()

Liveness, straight-line code

		in	out
1:	`(x2 <- rdi)`	(rdi)	(x2)
2:	`(x2 *= x2)`	(x2)	(x2)
3:	`(dx2 <- x2)`	(rdi x2)	(dx2 rdi)
4:	`(dx2 *= 2)`	(dx2 rdi)	(rdi)
5:	`(tx <- rdi)`	(dx2 rdi)	(dx2 tx)
6:	`(tx *= 3)`	(dx2 tx)	(dx2 tx)
7:	`(rax <- dx2)`	(dx2 tx)	(rax tx)
8:	`(rax += tx)`	(r12 r13 r14 r15 rax rbp rbx tx)	(r12 r13 r14 r15 rax rbp rbx)
9:	`(rax += 4)`	(r12 r13 r14 r15 rax rbp rbx)	(r12 r13 r14 r15 rax rbp rbx)
10:	`(return)`	(r12 r13 r14 r15 rax rbp rbx)	()

Liveness, straight-line code

		in	out
1:	`(x2 <- rdi)`	(rdi)	(x2)
2:	`(x2 *= x2)`	(x2)	(rdi x2)
3:	`(dx2 <- x2)`	(rdi x2)	(dx2 rdi)
4:	`(dx2 *= 2)`	(dx2 rdi)	(dx2 rdi)
5:	`(tx <- rdi)`	(dx2 rdi)	(dx2 tx)
6:	`(tx *= 3)`	(dx2 tx)	(dx2 tx)
7:	`(rax <- dx2)`	(dx2 tx)	(r12 r13 r14 r15 rax rbp rbx tx)
8:	`(rax += tx)`	(r12 r13 r14 r15 rax rbp rbx tx)	(r12 r13 r14 r15 rax rbp rbx)
9:	`(rax += 4)`	(r12 r13 r14 r15 rax rbp rbx)	(r12 r13 r14 r15 rax rbp rbx)
10:	`(return)`	(r12 r13 r14 r15 rax rbp rbx)	()

Liveness, straight-line code

		in	out
1:	`(x2 <- rdi)`	(rdi)	(x2)
2:	`(x2 *= x2)`	(rdi x2)	(rdi x2)
3:	`(dx2 <- x2)`	(rdi x2)	(dx2 rdi)
4:	`(dx2 *= 2)`	(dx2 rdi)	(dx2 rdi)
5:	`(tx <- rdi)`	(dx2 rdi)	(dx2 tx)
6:	`(tx *= 3)`	(dx2 tx)	(dx2 tx)
7:	`(rax <- dx2)`	(dx2 r12 r13 r14 r15 rbp rbx tx)	(r12 r13 r14 r15 rax rbp rbx tx)
8:	`(rax += tx)`	(r12 r13 r14 r15 rax rbp rbx tx)	(r12 r13 r14 r15 rax rbp rbx)
9:	`(rax += 4)`	(r12 r13 r14 r15 rax rbp rbx)	(r12 r13 r14 r15 rax rbp rbx)
10:	`(return)`	(r12 r13 r14 r15 rax rbp rbx)	()

Liveness, straight-line code

		in	out
1:	`(x2 <- rdi)`	(rdi)	(rdi x2)
2:	`(x2 *= x2)`	(rdi x2)	(rdi x2)
3:	`(dx2 <- x2)`	(rdi x2)	(dx2 rdi)
4:	`(dx2 *= 2)`	(dx2 rdi)	(dx2 rdi)
5:	`(tx <- rdi)`	(dx2 rdi)	(dx2 tx)
6:	`(tx *= 3)`	(dx2 tx)	(dx2 r12 r13 r14 r15 rbp rbx tx)
7:	`(rax <- dx2)`	(dx2 r12 r13 r14 r15 rbp rbx tx)	(r12 r13 r14 r15 rax rbp rbx tx)
8:	`(rax += tx)`	(r12 r13 r14 r15 rax rbp rbx tx)	(r12 r13 r14 r15 rax rbp rbx)
9:	`(rax += 4)`	(r12 r13 r14 r15 rax rbp rbx)	(r12 r13 r14 r15 rax rbp rbx)
10:	`(return)`	(r12 r13 r14 r15 rax rbp rbx)	()

Liveness, straight-line code

		in	out
1:	`(x2 <- rdi)`	(rdi)	(rdi x2)
2:	`(x2 *= x2)`	(rdi x2)	(rdi x2)
3:	`(dx2 <- x2)`	(rdi x2)	(dx2 rdi)
4:	`(dx2 *= 2)`	(dx2 rdi)	(dx2 rdi)
5:	`(tx <- rdi)`	(dx2 rdi)	(dx2 tx)
6:	`(tx *= 3)`	(dx2 r12 r13 r14 r15 rbp rbx tx)	(dx2 r12 r13 r14 r15 rbp rbx tx)
7:	`(rax <- dx2)`	(dx2 r12 r13 r14 r15 rbp rbx tx)	(r12 r13 r14 r15 rax rbp rbx tx)
8:	`(rax += tx)`	(r12 r13 r14 r15 rax rbp rbx tx)	(r12 r13 r14 r15 rax rbp rbx)
9:	`(rax += 4)`	(r12 r13 r14 r15 rax rbp rbx)	(r12 r13 r14 r15 rax rbp rbx)
10:	`(return)`	(r12 r13 r14 r15 rax rbp rbx)	()

Liveness, straight-line code

		in	out
1:	`(x2 <- rdi)`	(rdi)	(rdi x2)
2:	`(x2 *= x2)`	(rdi x2)	(rdi x2)
3:	`(dx2 <- x2)`	(rdi x2)	(dx2 rdi)
4:	`(dx2 *= 2)`	(dx2 rdi)	(dx2 rdi)
5:	`(tx <- rdi)`	(dx2 rdi)	(dx2 r12 r13 r14 r15 rbp rbx tx)
6:	`(tx *= 3)`	(dx2 r12 r13 r14 r15 rbp rbx tx)	(dx2 r12 r13 r14 r15 rbp rbx tx)
7:	`(rax <- dx2)`	(dx2 r12 r13 r14 r15 rbp rbx tx)	(r12 r13 r14 r15 rax rbp rbx tx)
8:	`(rax += tx)`	(r12 r13 r14 r15 rax rbp rbx tx)	(r12 r13 r14 r15 rax rbp rbx)
9:	`(rax += 4)`	(r12 r13 r14 r15 rax rbp rbx)	(r12 r13 r14 r15 rax rbp rbx)
10:	`(return)`	(r12 r13 r14 r15 rax rbp rbx)	()

Liveness, straight-line code

		in	out
1:	`(x2 <- rdi)`	(rdi)	(rdi x2)
2:	`(x2 *= x2)`	(rdi x2)	(rdi x2)
3:	`(dx2 <- x2)`	(rdi x2)	(dx2 rdi)
4:	`(dx2 *= 2)`	(dx2 rdi)	(dx2 rdi)
5:	`(tx <- rdi)`	(dx2 r12 r13 r14 r15 rbp rbx rdi)	(dx2 r12 r13 r14 r15 rbp rbx tx)
6:	`(tx *= 3)`	(dx2 r12 r13 r14 r15 rbp rbx tx)	(dx2 r12 r13 r14 r15 rbp rbx tx)
7:	`(rax <- dx2)`	(dx2 r12 r13 r14 r15 rbp rbx tx)	(r12 r13 r14 r15 rax rbp rbx tx)
8:	`(rax += tx)`	(r12 r13 r14 r15 rax rbp rbx tx)	(r12 r13 r14 r15 rax rbp rbx)
9:	`(rax += 4)`	(r12 r13 r14 r15 rax rbp rbx)	(r12 r13 r14 r15 rax rbp rbx)
10:	`(return)`	(r12 r13 r14 r15 rax rbp rbx)	()

Liveness, straight-line code

		in	out
1:	`(x2 <- rdi)`	(rdi)	(rdi x2)
2:	`(x2 *= x2)`	(rdi x2)	(rdi x2)
3:	`(dx2 <- x2)`	(rdi x2)	(dx2 rdi)
4:	`(dx2 *= 2)`	(dx2 rdi)	(dx2 r12 r13 r14 r15 rbp rbx rdi)
5:	`(tx <- rdi)`	(dx2 r12 r13 r14 r15 rbp rbx rdi)	(dx2 r12 r13 r14 r15 rbp rbx tx)
6:	`(tx *= 3)`	(dx2 r12 r13 r14 r15 rbp rbx tx)	(dx2 r12 r13 r14 r15 rbp rbx tx)
7:	`(rax <- dx2)`	(dx2 r12 r13 r14 r15 rbp rbx tx)	(r12 r13 r14 r15 rax rbp rbx tx)
8:	`(rax += tx)`	(r12 r13 r14 r15 rax rbp rbx tx)	(r12 r13 r14 r15 rax rbp rbx)
9:	`(rax += 4)`	(r12 r13 r14 r15 rax rbp rbx)	(r12 r13 r14 r15 rax rbp rbx)
10:	`(return)`	(r12 r13 r14 r15 rax rbp rbx)	()

Liveness, straight-line code

		in	out
1:	`(x2 <- rdi)`	(rdi)	(rdi x2)
2:	`(x2 *= x2)`	(rdi x2)	(rdi x2)
3:	`(dx2 <- x2)`	(rdi x2)	(dx2 rdi)
4:	`(dx2 *= 2)`	(dx2 r12 r13 r14 r15 rbp rbx rdi)	(dx2 r12 r13 r14 r15 rbp rbx rdi)
5:	`(tx <- rdi)`	(dx2 r12 r13 r14 r15 rbp rbx rdi)	(dx2 r12 r13 r14 r15 rbp rbx tx)
6:	`(tx *= 3)`	(dx2 r12 r13 r14 r15 rbp rbx tx)	(dx2 r12 r13 r14 r15 rbp rbx tx)
7:	`(rax <- dx2)`	(dx2 r12 r13 r14 r15 rbp rbx tx)	(r12 r13 r14 r15 rax rbp rbx tx)
8:	`(rax += tx)`	(r12 r13 r14 r15 rax rbp rbx tx)	(r12 r13 r14 r15 rax rbp rbx)
9:	`(rax += 4)`	(r12 r13 r14 r15 rax rbp rbx)	(r12 r13 r14 r15 rax rbp rbx)
10:	`(return)`	(r12 r13 r14 r15 rax rbp rbx)	()

Liveness, straight-line code

		in	out
1:	`(x2 <- rdi)`	(rdi)	(rdi x2)
2:	`(x2 *= x2)`	(rdi x2)	(rdi x2)
3:	`(dx2 <- x2)`	(rdi x2)	(dx2 r12 r13 r14 r15 rbp rbx rdi)
4:	`(dx2 *= 2)`	(dx2 r12 r13 r14 r15 rbp rbx rdi)	(dx2 r12 r13 r14 r15 rbp rbx rdi)
5:	`(tx <- rdi)`	(dx2 r12 r13 r14 r15 rbp rbx rdi)	(dx2 r12 r13 r14 r15 rbp rbx tx)
6:	`(tx *= 3)`	(dx2 r12 r13 r14 r15 rbp rbx tx)	(dx2 r12 r13 r14 r15 rbp rbx tx)
7:	`(rax <- dx2)`	(dx2 r12 r13 r14 r15 rbp rbx tx)	(r12 r13 r14 r15 rax rbp rbx tx)
8:	`(rax += tx)`	(r12 r13 r14 r15 rax rbp rbx tx)	(r12 r13 r14 r15 rax rbp rbx)
9:	`(rax += 4)`	(r12 r13 r14 r15 rax rbp rbx)	(r12 r13 r14 r15 rax rbp rbx)
10:	`(return)`	(r12 r13 r14 r15 rax rbp rbx)	()

Liveness, straight-line code

		in	out
1:	`(x2 <- rdi)`	(rdi)	(rdi x2)
2:	`(x2 *= x2)`	(rdi x2)	(rdi x2)
3:	`(dx2 <- x2)`	(r12 r13 r14 r15 rbp rbx rdi x2)	(dx2 r12 r13 r14 r15 rbp rbx rdi)
4:	`(dx2 *= 2)`	(dx2 r12 r13 r14 r15 rbp rbx rdi)	(dx2 r12 r13 r14 r15 rbp rbx rdi)
5:	`(tx <- rdi)`	(dx2 r12 r13 r14 r15 rbp rbx rdi)	(dx2 r12 r13 r14 r15 rbp rbx tx)
6:	`(tx *= 3)`	(dx2 r12 r13 r14 r15 rbp rbx tx)	(dx2 r12 r13 r14 r15 rbp rbx tx)
7:	`(rax <- dx2)`	(dx2 r12 r13 r14 r15 rbp rbx tx)	(r12 r13 r14 r15 rax rbp rbx tx)
8:	`(rax += tx)`	(r12 r13 r14 r15 rax rbp rbx tx)	(r12 r13 r14 r15 rax rbp rbx)
9:	`(rax += 4)`	(r12 r13 r14 r15 rax rbp rbx)	(r12 r13 r14 r15 rax rbp rbx)
10:	`(return)`	(r12 r13 r14 r15 rax rbp rbx)	()

Liveness, straight-line code

		in	out
1:	`(x2 <- rdi)`	(rdi)	(rdi x2)
2:	`(x2 *= x2)`	(rdi x2)	(r12 r13 r14 r15 rbp rbx rdi x2)
3:	`(dx2 <- x2)`	(r12 r13 r14 r15 rbp rbx rdi x2)	(dx2 r12 r13 r14 r15 rbp rbx rdi)
4:	`(dx2 *= 2)`	(dx2 r12 r13 r14 r15 rbp rbx rdi)	(dx2 r12 r13 r14 r15 rbp rbx rdi)
5:	`(tx <- rdi)`	(dx2 r12 r13 r14 r15 rbp rbx rdi)	(dx2 r12 r13 r14 r15 rbp rbx tx)
6:	`(tx *= 3)`	(dx2 r12 r13 r14 r15 rbp rbx tx)	(dx2 r12 r13 r14 r15 rbp rbx tx)
7:	`(rax <- dx2)`	(dx2 r12 r13 r14 r15 rbp rbx tx)	(r12 r13 r14 r15 rax rbp rbx tx)
8:	`(rax += tx)`	(r12 r13 r14 r15 rax rbp rbx tx)	(r12 r13 r14 r15 rax rbp rbx)
9:	`(rax += 4)`	(r12 r13 r14 r15 rax rbp rbx)	(r12 r13 r14 r15 rax rbp rbx)
10:	`(return)`	(r12 r13 r14 r15 rax rbp rbx)	()

Liveness, straight-line code

		in	out
1:	`(x2 <- rdi)`	(rdi)	(rdi x2)
2:	`(x2 *= x2)`	(r12 r13 r14 r15 rbp rbx rdi x2)	(r12 r13 r14 r15 rbp rbx rdi x2)
3:	`(dx2 <- x2)`	(r12 r13 r14 r15 rbp rbx rdi x2)	(dx2 r12 r13 r14 r15 rbp rbx rdi)
4:	`(dx2 *= 2)`	(dx2 r12 r13 r14 r15 rbp rbx rdi)	(dx2 r12 r13 r14 r15 rbp rbx rdi)
5:	`(tx <- rdi)`	(dx2 r12 r13 r14 r15 rbp rbx rdi)	(dx2 r12 r13 r14 r15 rbp rbx tx)
6:	`(tx *= 3)`	(dx2 r12 r13 r14 r15 rbp rbx tx)	(dx2 r12 r13 r14 r15 rbp rbx tx)
7:	`(rax <- dx2)`	(dx2 r12 r13 r14 r15 rbp rbx tx)	(r12 r13 r14 r15 rax rbp rbx tx)
8:	`(rax += tx)`	(r12 r13 r14 r15 rax rbp rbx tx)	(r12 r13 r14 r15 rax rbp rbx)
9:	`(rax += 4)`	(r12 r13 r14 r15 rax rbp rbx)	(r12 r13 r14 r15 rax rbp rbx)
10:	`(return)`	(r12 r13 r14 r15 rax rbp rbx)	()

Liveness, straight-line code

		in	out
1:	`(x2 <- rdi)`	(rdi)	(r12 r13 r14 r15 rbp rbx rdi x2)
2:	`(x2 *= x2)`	(r12 r13 r14 r15 rbp rbx rdi x2)	(r12 r13 r14 r15 rbp rbx rdi x2)
3:	`(dx2 <- x2)`	(r12 r13 r14 r15 rbp rbx rdi x2)	(dx2 r12 r13 r14 r15 rbp rbx rdi)
4:	`(dx2 *= 2)`	(dx2 r12 r13 r14 r15 rbp rbx rdi)	(dx2 r12 r13 r14 r15 rbp rbx rdi)
5:	`(tx <- rdi)`	(dx2 r12 r13 r14 r15 rbp rbx rdi)	(dx2 r12 r13 r14 r15 rbp rbx tx)
6:	`(tx *= 3)`	(dx2 r12 r13 r14 r15 rbp rbx tx)	(dx2 r12 r13 r14 r15 rbp rbx tx)
7:	`(rax <- dx2)`	(dx2 r12 r13 r14 r15 rbp rbx tx)	(r12 r13 r14 r15 rax rbp rbx tx)
8:	`(rax += tx)`	(r12 r13 r14 r15 rax rbp rbx tx)	(r12 r13 r14 r15 rax rbp rbx)
9:	`(rax += 4)`	(r12 r13 r14 r15 rax rbp rbx)	(r12 r13 r14 r15 rax rbp rbx)
10:	`(return)`	(r12 r13 r14 r15 rax rbp rbx)	()

Liveness, straight-line code

		in	out
1:	`(x2 <- rdi)`	(r12 r13 r14 r15 rbp rbx rdi)	(r12 r13 r14 r15 rbp rbx rdi x2)
2:	`(x2 *= x2)`	(r12 r13 r14 r15 rbp rbx rdi x2)	(r12 r13 r14 r15 rbp rbx rdi x2)
3:	`(dx2 <- x2)`	(r12 r13 r14 r15 rbp rbx rdi x2)	(dx2 r12 r13 r14 r15 rbp rbx rdi)
4:	`(dx2 *= 2)`	(dx2 r12 r13 r14 r15 rbp rbx rdi)	(dx2 r12 r13 r14 r15 rbp rbx rdi)
5:	`(tx <- rdi)`	(dx2 r12 r13 r14 r15 rbp rbx rdi)	(dx2 r12 r13 r14 r15 rbp rbx tx)
6:	`(tx *= 3)`	(dx2 r12 r13 r14 r15 rbp rbx tx)	(dx2 r12 r13 r14 r15 rbp rbx tx)
7:	`(rax <- dx2)`	(dx2 r12 r13 r14 r15 rbp rbx tx)	(r12 r13 r14 r15 rax rbp rbx tx)
8:	`(rax += tx)`	(r12 r13 r14 r15 rax rbp rbx tx)	(r12 r13 r14 r15 rax rbp rbx)
9:	`(rax += 4)`	(r12 r13 r14 r15 rax rbp rbx)	(r12 r13 r14 r15 rax rbp rbx)
10:	`(return)`	(r12 r13 r14 r15 rax rbp rbx)	()

Liveness with a loop

		in	out
1:	`(u <- 0)`	()	()
2:	`(d <- rdi)`	()	()
3:	`:top`	()	()
4:	`(cjump d = 0 :z :nz)`	()	()
5:	`:z`	()	()
6:	`(rax <- u)`	()	()
7:	`(return)`	()	()
8:	`:nz`	()	()
9:	`(u += 1)`	()	()
10:	`(d -= 1)`	()	()
11:	`(goto :top)`	()	()

Liveness with a loop

		in	out
1:	`(u <- 0)`	()	()
2:	`(d <- rdi)`	(rdi)	()
3:	`:top`	()	()
4:	`(cjump d = 0 :z :nz)`	(d)	()
5:	`:z`	()	()
6:	`(rax <- u)`	(u)	()
7:	`(return)`	(r12 r13 r14 r15 rax rbp rbx)	()
8:	`:nz`	()	()
9:	`(u += 1)`	(u)	()
10:	`(d -= 1)`	(d)	()
11:	`(goto :top)`	()	()

Liveness with a loop

		in	out
1:	`(u <- 0)`	()	(rdi)
2:	`(d <- rdi)`	(rdi)	()
3:	`:top`	()	(d)
4:	`(cjump d = 0 :z :nz)`	(d)	()
5:	`:z`	()	(u)
6:	`(rax <- u)`	(u)	(r12 r13 r14 r15 rax rbp rbx)
7:	`(return)`	(r12 r13 r14 r15 rax rbp rbx)	()
8:	`:nz`	()	(u)
9:	`(u += 1)`	(u)	(d)
10:	`(d -= 1)`	(d)	()
11:	`(goto :top)`	()	()

Liveness with a loop

		in	out
1:	`(u <- 0)`	(rdi)	(rdi)
2:	`(d <- rdi)`	(rdi)	()
3:	`:top`	(d)	(d)
4:	`(cjump d = 0 :z :nz)`	(d)	()
5:	`:z`	(u)	(u)
6:	`(rax <- u)`	(r12 r13 r14 r15 rbp rbx u)	(r12 r13 r14 r15 rax rbp rbx)
7:	`(return)`	(r12 r13 r14 r15 rax rbp rbx)	()
8:	`:nz`	(u)	(u)
9:	`(u += 1)`	(d u)	(d)
10:	`(d -= 1)`	(d)	()
11:	`(goto :top)`	()	()

Liveness with a loop

		in	out
1:	`(u <- 0)`	(rdi)	(rdi)
2:	`(d <- rdi)`	(rdi)	(d)
3:	`:top`	(d)	(d)
4:	`(cjump d = 0 :z :nz)`	(d)	(u)
5:	`:z`	(u)	(r12 r13 r14 r15 rbp rbx u)
6:	`(rax <- u)`	(r12 r13 r14 r15 rbp rbx u)	(r12 r13 r14 r15 rax rbp rbx)
7:	`(return)`	(r12 r13 r14 r15 rax rbp rbx)	()
8:	`:nz`	(u)	(d u)
9:	`(u += 1)`	(d u)	(d)
10:	`(d -= 1)`	(d)	()
11:	`(goto :top)`	()	(d)

Liveness with a loop

		in	out
1:	`(u <- 0)`	(rdi)	(rdi)
2:	`(d <- rdi)`	(rdi)	(d)
3:	`:top`	(d)	(d)
4:	`(cjump d = 0 :z :nz)`	(d u)	(u)
5:	`:z`	(r12 r13 r14 r15 rbp rbx u)	(r12 r13 r14 r15 rbp rbx u)
6:	`(rax <- u)`	(r12 r13 r14 r15 rbp rbx u)	(r12 r13 r14 r15 rax rbp rbx)
7:	`(return)`	(r12 r13 r14 r15 rax rbp rbx)	()
8:	`:nz`	(d u)	(d u)
9:	`(u += 1)`	(d u)	(d)
10:	`(d -= 1)`	(d)	()
11:	`(goto :top)`	(d)	(d)

Liveness with a loop

		in	out
1:	`(u <- 0)`	(rdi)	(rdi)
2:	`(d <- rdi)`	(rdi)	(d)
3:	`:top`	(d)	(d u)
4:	`(cjump d = 0 :z :nz)`	(d u)	(d r12 r13 r14 r15 rbp rbx u)
5:	`:z`	(r12 r13 r14 r15 rbp rbx u)	(r12 r13 r14 r15 rbp rbx u)
6:	`(rax <- u)`	(r12 r13 r14 r15 rbp rbx u)	(r12 r13 r14 r15 rax rbp rbx)
7:	`(return)`	(r12 r13 r14 r15 rax rbp rbx)	()
8:	`:nz`	(d u)	(d u)
9:	`(u += 1)`	(d u)	(d)
10:	`(d -= 1)`	(d)	(d)
11:	`(goto :top)`	(d)	(d)

Liveness with a loop

		in	out
1:	`(u <- 0)`	(rdi)	(rdi)
2:	`(d <- rdi)`	(rdi)	(d)
3:	`:top`	(d u)	(d u)
4:	`(cjump d = 0 :z :nz)`	(d r12 r13 r14 r15 rbp rbx u)	(d r12 r13 r14 r15 rbp rbx u)
5:	`:z`	(r12 r13 r14 r15 rbp rbx u)	(r12 r13 r14 r15 rbp rbx u)
6:	`(rax <- u)`	(r12 r13 r14 r15 rbp rbx u)	(r12 r13 r14 r15 rax rbp rbx)
7:	`(return)`	(r12 r13 r14 r15 rax rbp rbx)	()
8:	`:nz`	(d u)	(d u)
9:	`(u += 1)`	(d u)	(d)
10:	`(d -= 1)`	(d)	(d)
11:	`(goto :top)`	(d)	(d)

Liveness with a loop

		in	out
1:	`(u <- 0)`	(rdi)	(rdi)
2:	`(d <- rdi)`	(rdi)	(d u)
3:	`:top`	(d u)	(d r12 r13 r14 r15 rbp rbx u)
4:	`(cjump d = 0 :z :nz)`	(d r12 r13 r14 r15 rbp rbx u)	(d r12 r13 r14 r15 rbp rbx u)
5:	`:z`	(r12 r13 r14 r15 rbp rbx u)	(r12 r13 r14 r15 rbp rbx u)
6:	`(rax <- u)`	(r12 r13 r14 r15 rbp rbx u)	(r12 r13 r14 r15 rax rbp rbx)
7:	`(return)`	(r12 r13 r14 r15 rax rbp rbx)	()
8:	`:nz`	(d u)	(d u)
9:	`(u += 1)`	(d u)	(d)
10:	`(d -= 1)`	(d)	(d)
11:	`(goto :top)`	(d)	(d u)

Liveness with a loop

		in	out
1:	`(u <- 0)`	(rdi)	(rdi)
2:	`(d <- rdi)`	(rdi u)	(d u)
3:	`:top`	(d r12 r13 r14 r15 rbp rbx u)	(d r12 r13 r14 r15 rbp rbx u)
4:	`(cjump d = 0 :z :nz)`	(d r12 r13 r14 r15 rbp rbx u)	(d r12 r13 r14 r15 rbp rbx u)
5:	`:z`	(r12 r13 r14 r15 rbp rbx u)	(r12 r13 r14 r15 rbp rbx u)
6:	`(rax <- u)`	(r12 r13 r14 r15 rbp rbx u)	(r12 r13 r14 r15 rax rbp rbx)
7:	`(return)`	(r12 r13 r14 r15 rax rbp rbx)	()
8:	`:nz`	(d u)	(d u)
9:	`(u += 1)`	(d u)	(d)
10:	`(d -= 1)`	(d)	(d)
11:	`(goto :top)`	(d u)	(d u)

Liveness with a loop

		in	out
1:	`(u <- 0)`	(rdi)	(rdi u)
2:	`(d <- rdi)`	(rdi u)	(d r12 r13 r14 r15 rbp rbx u)
3:	`:top`	(d r12 r13 r14 r15 rbp rbx u)	(d r12 r13 r14 r15 rbp rbx u)
4:	`(cjump d = 0 :z :nz)`	(d r12 r13 r14 r15 rbp rbx u)	(d r12 r13 r14 r15 rbp rbx u)
5:	`:z`	(r12 r13 r14 r15 rbp rbx u)	(r12 r13 r14 r15 rbp rbx u)
6:	`(rax <- u)`	(r12 r13 r14 r15 rbp rbx u)	(r12 r13 r14 r15 rax rbp rbx)
7:	`(return)`	(r12 r13 r14 r15 rax rbp rbx)	()
8:	`:nz`	(d u)	(d u)
9:	`(u += 1)`	(d u)	(d)
10:	`(d -= 1)`	(d)	(d u)
11:	`(goto :top)`	(d u)	(d r12 r13 r14 r15 rbp rbx u)

Liveness with a loop

		in	out
1:	`(u <- 0)`	(rdi)	(rdi u)
2:	`(d <- rdi)`	(r12 r13 r14 r15 rbp rbx rdi u)	(d r12 r13 r14 r15 rbp rbx u)
3:	`:top`	(d r12 r13 r14 r15 rbp rbx u)	(d r12 r13 r14 r15 rbp rbx u)
4:	`(cjump d = 0 :z :nz)`	(d r12 r13 r14 r15 rbp rbx u)	(d r12 r13 r14 r15 rbp rbx u)
5:	`:z`	(r12 r13 r14 r15 rbp rbx u)	(r12 r13 r14 r15 rbp rbx u)
6:	`(rax <- u)`	(r12 r13 r14 r15 rbp rbx u)	(r12 r13 r14 r15 rax rbp rbx)
7:	`(return)`	(r12 r13 r14 r15 rax rbp rbx)	()
8:	`:nz`	(d u)	(d u)
9:	`(u += 1)`	(d u)	(d)
10:	`(d -= 1)`	(d u)	(d u)
11:	`(goto :top)`	(d r12 r13 r14 r15 rbp rbx u)	(d r12 r13 r14 r15 rbp rbx u)

Liveness with a loop

		in	out
1:	`(u <- 0)`	(rdi)	(r12 r13 r14 r15 rbp rbx rdi u)
2:	`(d <- rdi)`	(r12 r13 r14 r15 rbp rbx rdi u)	(d r12 r13 r14 r15 rbp rbx u)
3:	`:top`	(d r12 r13 r14 r15 rbp rbx u)	(d r12 r13 r14 r15 rbp rbx u)
4:	`(cjump d = 0 :z :nz)`	(d r12 r13 r14 r15 rbp rbx u)	(d r12 r13 r14 r15 rbp rbx u)
5:	`:z`	(r12 r13 r14 r15 rbp rbx u)	(r12 r13 r14 r15 rbp rbx u)
6:	`(rax <- u)`	(r12 r13 r14 r15 rbp rbx u)	(r12 r13 r14 r15 rax rbp rbx)
7:	`(return)`	(r12 r13 r14 r15 rax rbp rbx)	()
8:	`:nz`	(d u)	(d u)
9:	`(u += 1)`	(d u)	(d u)
10:	`(d -= 1)`	(d u)	(d r12 r13 r14 r15 rbp rbx u)
11:	`(goto :top)`	(d r12 r13 r14 r15 rbp rbx u)	(d r12 r13 r14 r15 rbp rbx u)

Liveness with a loop

		in	out
1:	`(u <- 0)`	(r12 r13 r14 r15 rbp rbx rdi)	(r12 r13 r14 r15 rbp rbx rdi u)
2:	`(d <- rdi)`	(r12 r13 r14 r15 rbp rbx rdi u)	(d r12 r13 r14 r15 rbp rbx u)
3:	`:top`	(d r12 r13 r14 r15 rbp rbx u)	(d r12 r13 r14 r15 rbp rbx u)
4:	`(cjump d = 0 :z :nz)`	(d r12 r13 r14 r15 rbp rbx u)	(d r12 r13 r14 r15 rbp rbx u)
5:	`:z`	(r12 r13 r14 r15 rbp rbx u)	(r12 r13 r14 r15 rbp rbx u)
6:	`(rax <- u)`	(r12 r13 r14 r15 rbp rbx u)	(r12 r13 r14 r15 rax rbp rbx)
7:	`(return)`	(r12 r13 r14 r15 rax rbp rbx)	()
8:	`:nz`	(d u)	(d u)
9:	`(u += 1)`	(d u)	(d u)
10:	`(d -= 1)`	(d r12 r13 r14 r15 rbp rbx u)	(d r12 r13 r14 r15 rbp rbx u)
11:	`(goto :top)`	(d r12 r13 r14 r15 rbp rbx u)	(d r12 r13 r14 r15 rbp rbx u)

Liveness with a loop

		in	out
1:	`(u <- 0)`	(r12 r13 r14 r15 rbp rbx rdi)	(r12 r13 r14 r15 rbp rbx rdi u)
2:	`(d <- rdi)`	(r12 r13 r14 r15 rbp rbx rdi u)	(d r12 r13 r14 r15 rbp rbx u)
3:	`:top`	(d r12 r13 r14 r15 rbp rbx u)	(d r12 r13 r14 r15 rbp rbx u)
4:	`(cjump d = 0 :z :nz)`	(d r12 r13 r14 r15 rbp rbx u)	(d r12 r13 r14 r15 rbp rbx u)
5:	`:z`	(r12 r13 r14 r15 rbp rbx u)	(r12 r13 r14 r15 rbp rbx u)
6:	`(rax <- u)`	(r12 r13 r14 r15 rbp rbx u)	(r12 r13 r14 r15 rax rbp rbx)
7:	`(return)`	(r12 r13 r14 r15 rax rbp rbx)	()
8:	`:nz`	(d u)	(d u)
9:	`(u += 1)`	(d u)	(d r12 r13 r14 r15 rbp rbx u)
10:	`(d -= 1)`	(d r12 r13 r14 r15 rbp rbx u)	(d r12 r13 r14 r15 rbp rbx u)
11:	`(goto :top)`	(d r12 r13 r14 r15 rbp rbx u)	(d r12 r13 r14 r15 rbp rbx u)

Liveness with a loop

		in	out
1:	`(u <- 0)`	(r12 r13 r14 r15 rbp rbx rdi)	(r12 r13 r14 r15 rbp rbx rdi u)
2:	`(d <- rdi)`	(r12 r13 r14 r15 rbp rbx rdi u)	(d r12 r13 r14 r15 rbp rbx u)
3:	`:top`	(d r12 r13 r14 r15 rbp rbx u)	(d r12 r13 r14 r15 rbp rbx u)
4:	`(cjump d = 0 :z :nz)`	(d r12 r13 r14 r15 rbp rbx u)	(d r12 r13 r14 r15 rbp rbx u)
5:	`:z`	(r12 r13 r14 r15 rbp rbx u)	(r12 r13 r14 r15 rbp rbx u)
6:	`(rax <- u)`	(r12 r13 r14 r15 rbp rbx u)	(r12 r13 r14 r15 rax rbp rbx)
7:	`(return)`	(r12 r13 r14 r15 rax rbp rbx)	()
8:	`:nz`	(d u)	(d u)
9:	`(u += 1)`	(d r12 r13 r14 r15 rbp rbx u)	(d r12 r13 r14 r15 rbp rbx u)
10:	`(d -= 1)`	(d r12 r13 r14 r15 rbp rbx u)	(d r12 r13 r14 r15 rbp rbx u)
11:	`(goto :top)`	(d r12 r13 r14 r15 rbp rbx u)	(d r12 r13 r14 r15 rbp rbx u)

Liveness with a loop

		in	out
1:	`(u <- 0)`	(r12 r13 r14 r15 rbp rbx rdi)	(r12 r13 r14 r15 rbp rbx rdi u)
2:	`(d <- rdi)`	(r12 r13 r14 r15 rbp rbx rdi u)	(d r12 r13 r14 r15 rbp rbx u)
3:	`:top`	(d r12 r13 r14 r15 rbp rbx u)	(d r12 r13 r14 r15 rbp rbx u)
4:	`(cjump d = 0 :z :nz)`	(d r12 r13 r14 r15 rbp rbx u)	(d r12 r13 r14 r15 rbp rbx u)
5:	`:z`	(r12 r13 r14 r15 rbp rbx u)	(r12 r13 r14 r15 rbp rbx u)
6:	`(rax <- u)`	(r12 r13 r14 r15 rbp rbx u)	(r12 r13 r14 r15 rax rbp rbx)
7:	`(return)`	(r12 r13 r14 r15 rax rbp rbx)	()
8:	`:nz`	(d u)	(d r12 r13 r14 r15 rbp rbx u)
9:	`(u += 1)`	(d r12 r13 r14 r15 rbp rbx u)	(d r12 r13 r14 r15 rbp rbx u)
10:	`(d -= 1)`	(d r12 r13 r14 r15 rbp rbx u)	(d r12 r13 r14 r15 rbp rbx u)
11:	`(goto :top)`	(d r12 r13 r14 r15 rbp rbx u)	(d r12 r13 r14 r15 rbp rbx u)

Liveness with a loop

		in	out
1:	`(u <- 0)`	(r12 r13 r14 r15 rbp rbx rdi)	(r12 r13 r14 r15 rbp rbx rdi u)
2:	`(d <- rdi)`	(r12 r13 r14 r15 rbp rbx rdi u)	(d r12 r13 r14 r15 rbp rbx u)
3:	`:top`	(d r12 r13 r14 r15 rbp rbx u)	(d r12 r13 r14 r15 rbp rbx u)
4:	`(cjump d = 0 :z :nz)`	(d r12 r13 r14 r15 rbp rbx u)	(d r12 r13 r14 r15 rbp rbx u)
5:	`:z`	(r12 r13 r14 r15 rbp rbx u)	(r12 r13 r14 r15 rbp rbx u)
6:	`(rax <- u)`	(r12 r13 r14 r15 rbp rbx u)	(r12 r13 r14 r15 rax rbp rbx)
7:	`(return)`	(r12 r13 r14 r15 rax rbp rbx)	()
8:	`:nz`	(d r12 r13 r14 r15 rbp rbx u)	(d r12 r13 r14 r15 rbp rbx u)
9:	`(u += 1)`	(d r12 r13 r14 r15 rbp rbx u)	(d r12 r13 r14 r15 rbp rbx u)
10:	`(d -= 1)`	(d r12 r13 r14 r15 rbp rbx u)	(d r12 r13 r14 r15 rbp rbx u)
11:	`(goto :top)`	(d r12 r13 r14 r15 rbp rbx u)	(d r12 r13 r14 r15 rbp rbx u)

Liveness infinite loop

		in	out
1:	`:top`	()	()
2:	`(x <- 1)`	()	()
3:	`(x += 1)`	()	()
4:	`(goto :top)`	()	()

Liveness infinite loop

		in	out
1:	`:top`	()	()
2:	`(x <- 1)`	()	()
3:	`(x += 1)`	(x)	()
4:	`(goto :top)`	()	()

Liveness infinite loop

		in	out
1:	`:top`	()	()
2:	`(x <- 1)`	()	(x)
3:	`(x += 1)`	(x)	()
4:	`(goto :top)`	()	()

Graph coloring

Build interference graph from the liveness information

- For each instruction:
  - Two variables live at the same time interfere with each other
  - Killed variables interfere with variables in the out set
  - Except that the variables `x` and `y` do not interfere if the instruction was `(x <- y)`
- All real registers interfere with each other
- Handle constrained arithmetic via extra edges

Live at the same time

What counts as live "at the same time"?

- We could say that two variables are live at the same time if they appear in either the 'in' or the 'out' set together for any instruction
- But, not counting instructions that have no predeccessors, every variable pair that is in an 'in' set, is also in an 'out' set
- Also, the only instruction that has no precedessor that actually gets run is the first one

So, live at the same time means appearing together in an 'out' set, or appearing together in the first instruction's 'in' set

Killed & Out Set

```
(:f 0 0
    (rax <- 2)
    (x <- 1)
    (return))
```

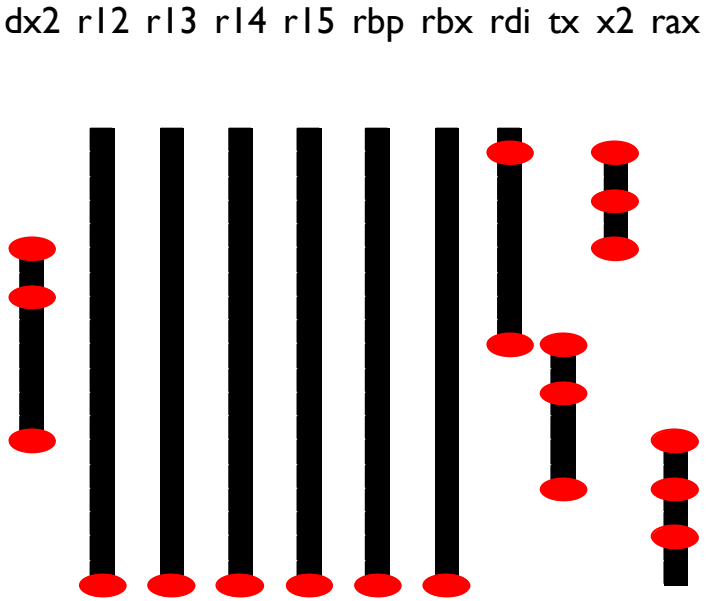
In this code, **x** has an empty live range, so it doesn't interfere with anything according to that clause of the definition. If we assign **x** to **rax**, however, we break the program!

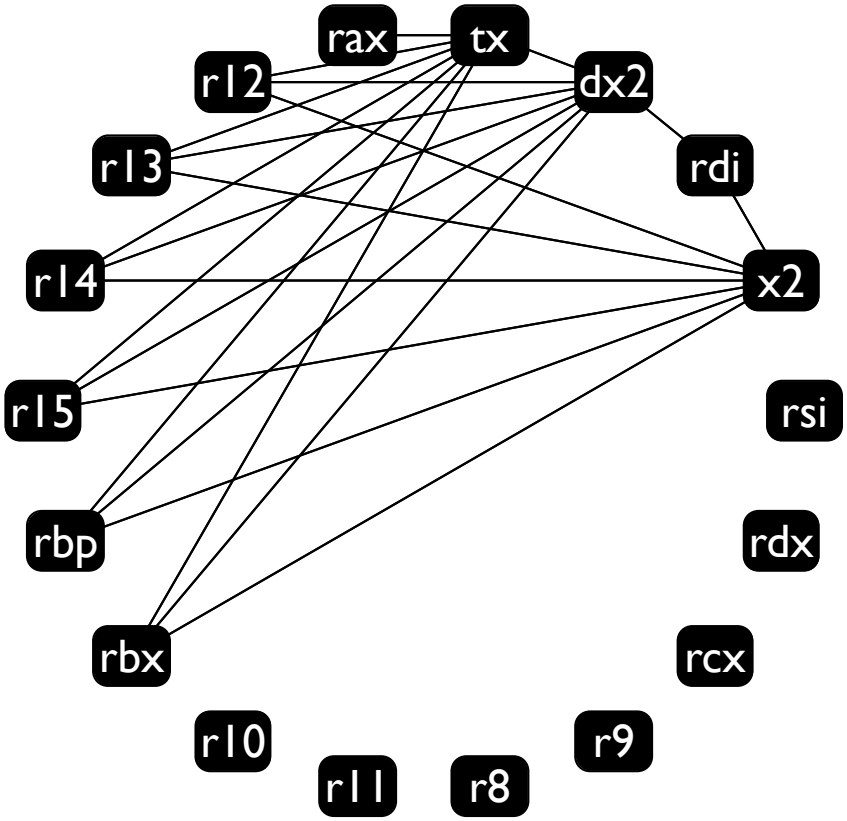
But, **x** is in the kill set of the second instruction and **rax** is live there, so we add an interference edge and thus don't break the program.

For each pair of x2, 2x2, 3x, and rax:

- do they interfere?;
- should they?

```
(:f
 1 0
 (x2 <- rdi)
 (x2 *= x2)
 (dx2 <- x2)
 (dx2 *= 2)
 (tx <- rdi)
 (tx *= 3)
 (rax <- dx2)
 (rax += tx)
 (rax += 4)
 (return))
```



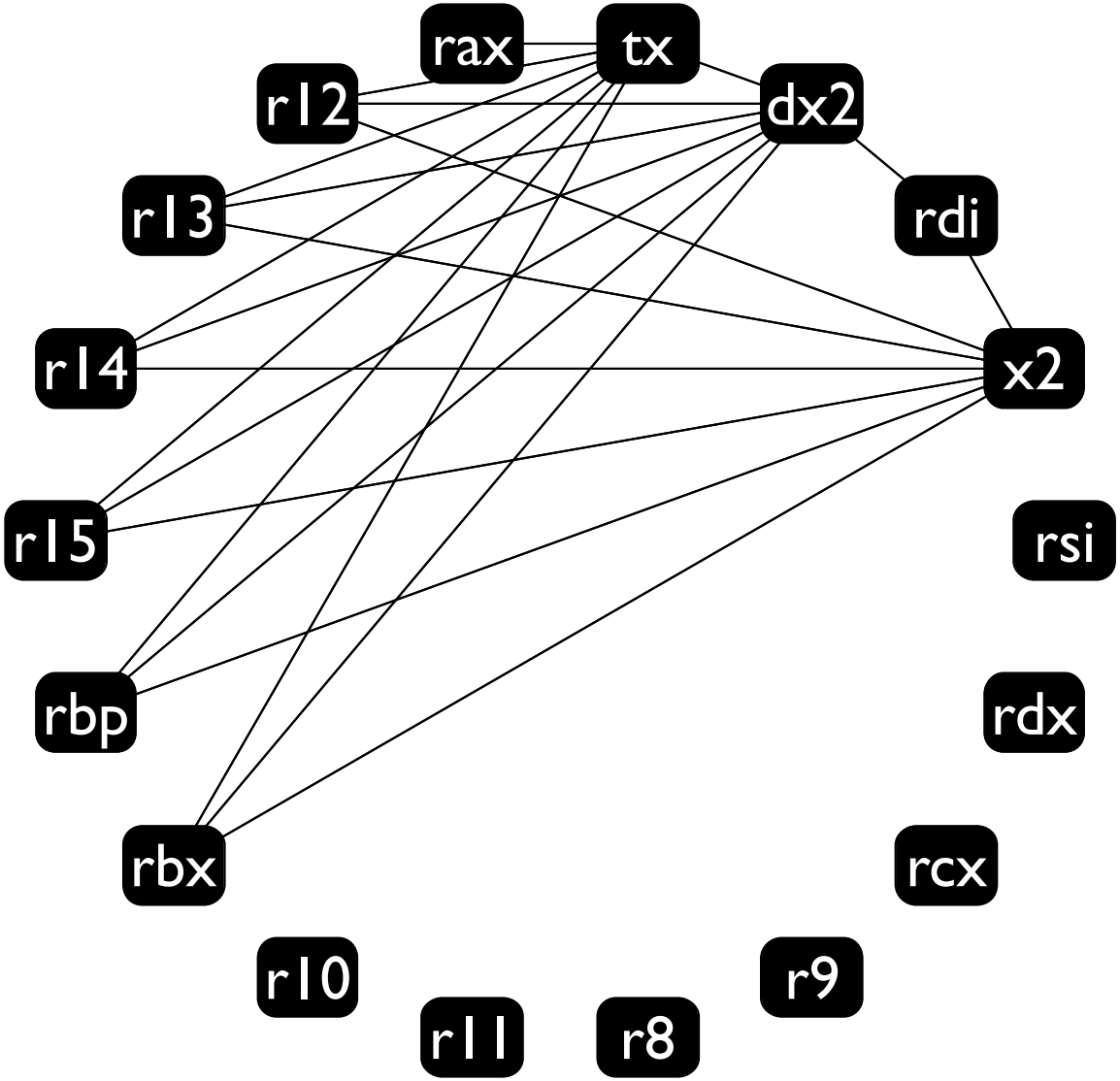


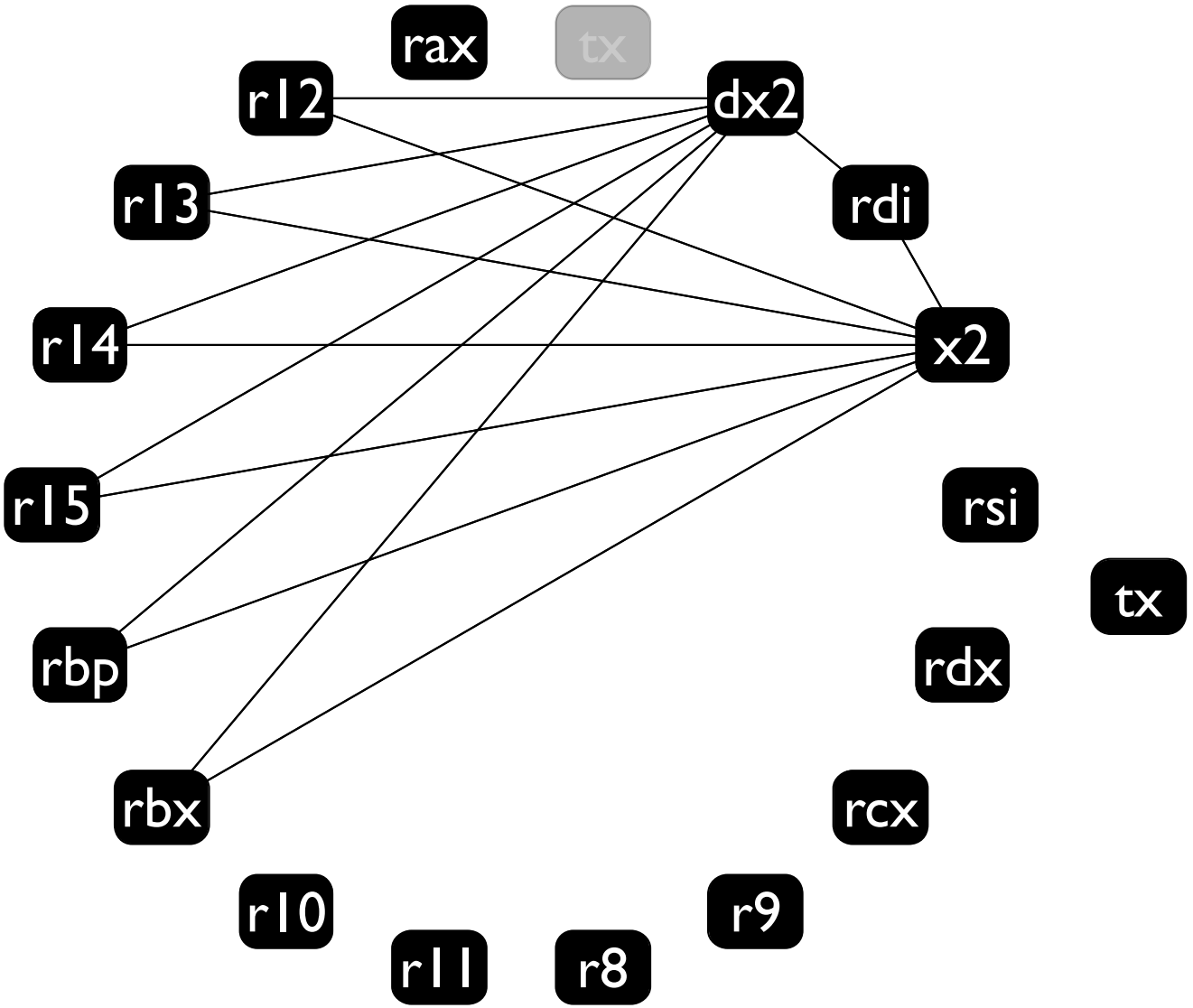
Constrained arithmetic operators

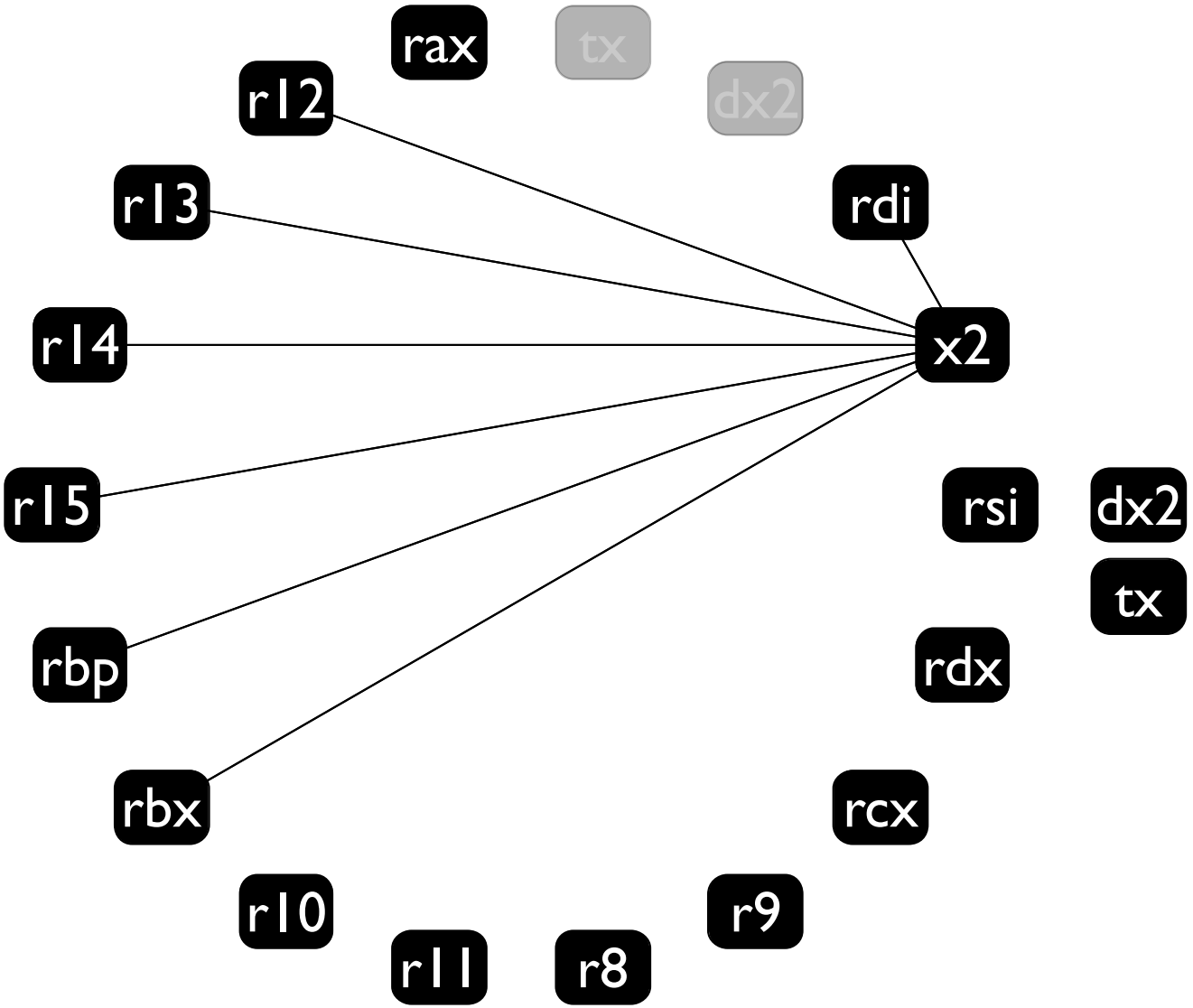
The `(x sop= sx)` instruction in L1 is limited to only shifting by the value of `rcx` (or by a constant in the other form).

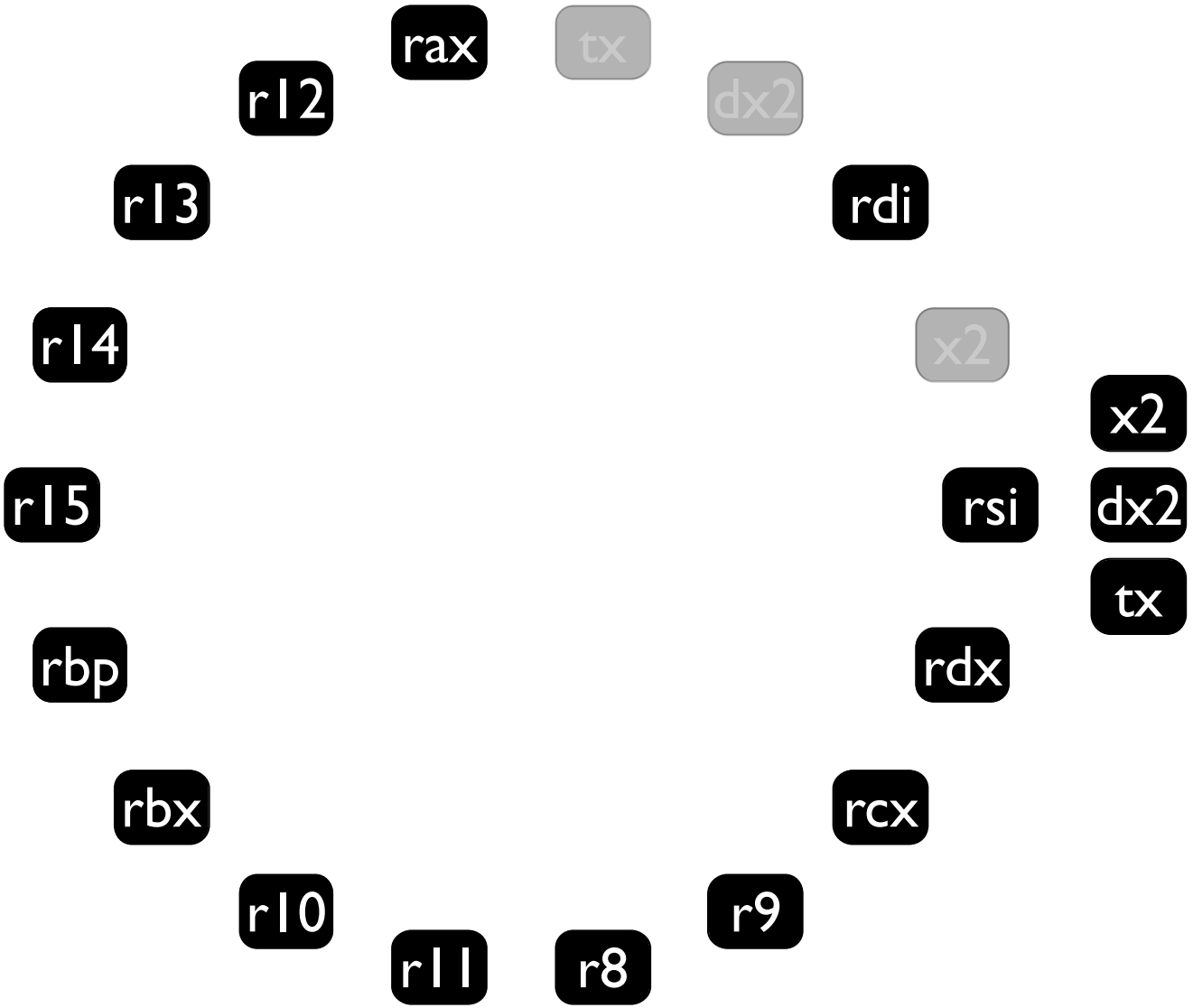
Add interference edges to disallow the illegal registers when building the interference graph, before starting the coloring.

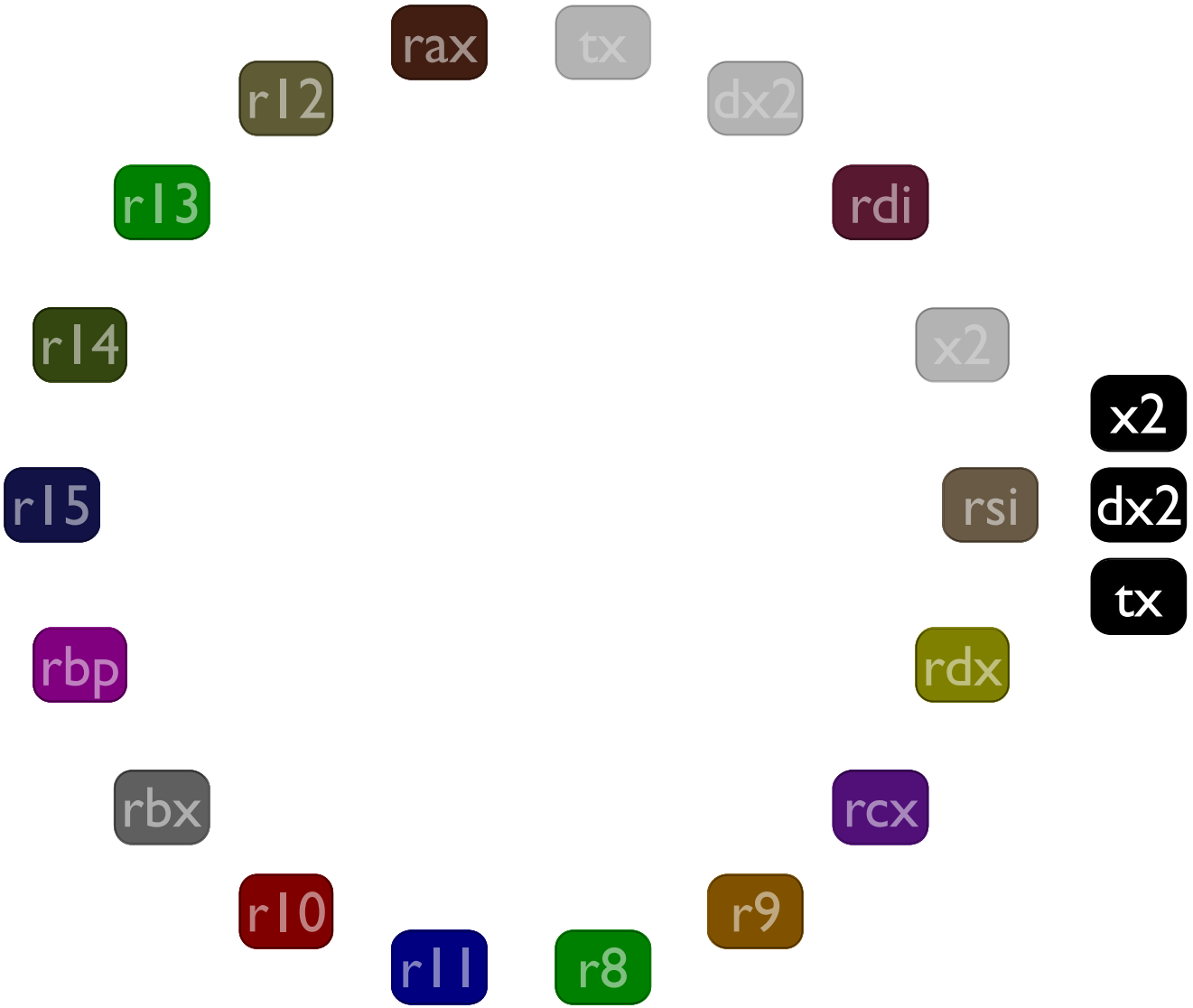
E.g., if you have this instruction `(a sop= b)` then add edges between `b` and every register except `rcx`, ensuring `b` ends up in `rcx` (or spilled).

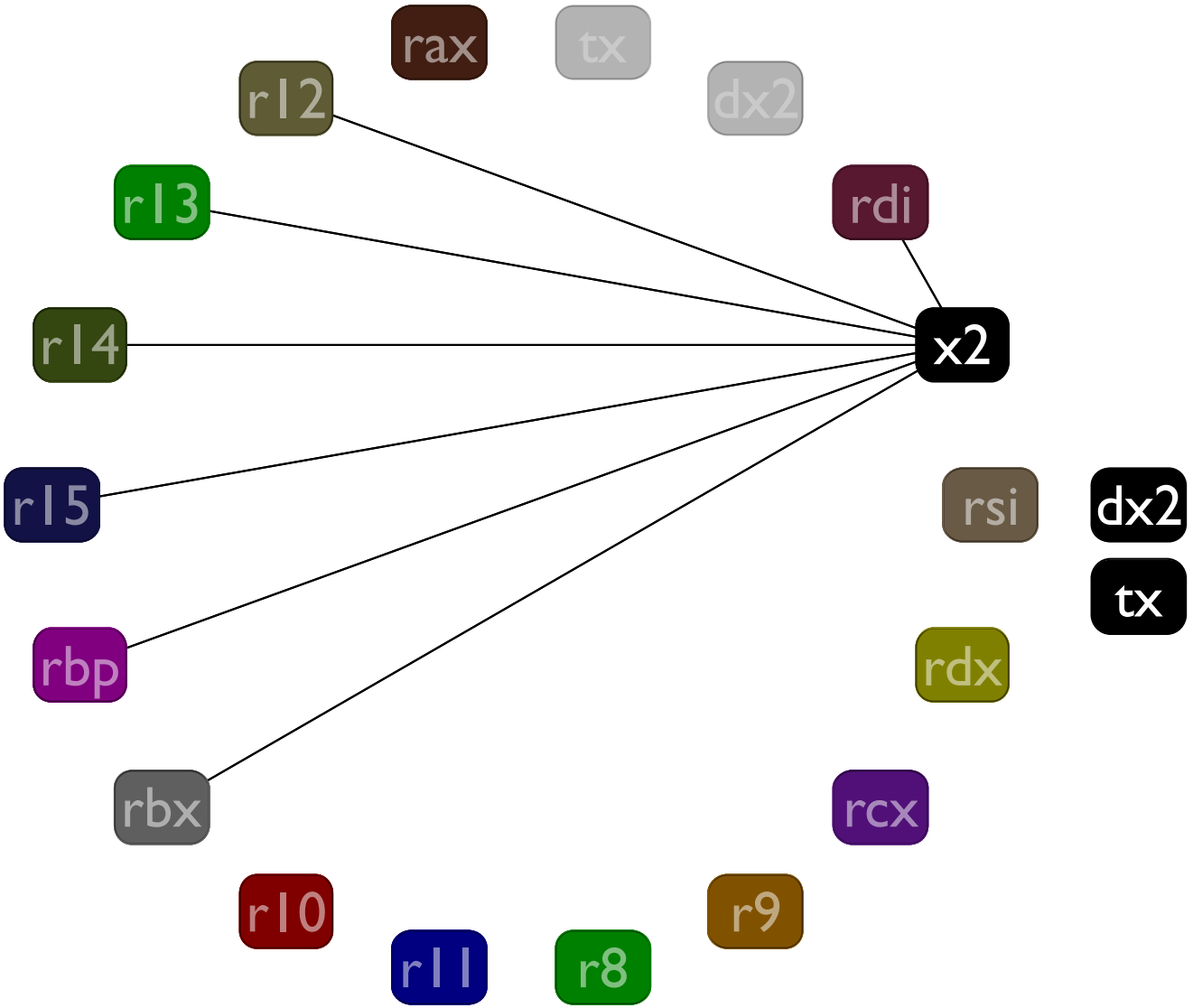

rax
tx
r12
dx2
r13
rdi
r14
x2
r15
rsi
rbp
rdx
rbx
rcx
r10
r9
r11
r8

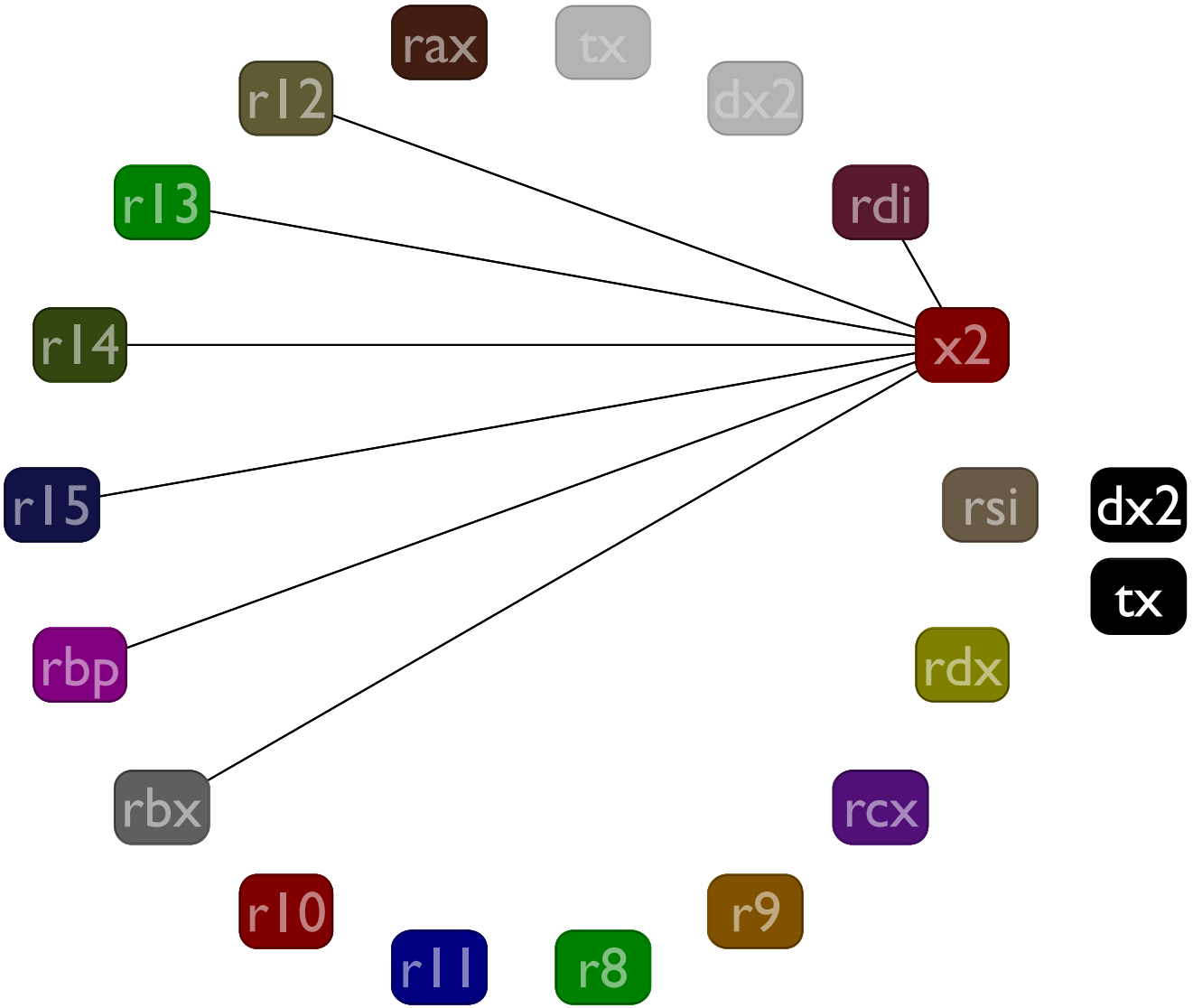

rax
tx
r12
dx2
r13
rdi
r14
x2
r15
rsi
tx
rbp
rdx
rbx
rcx
r10
r9
r11
r8

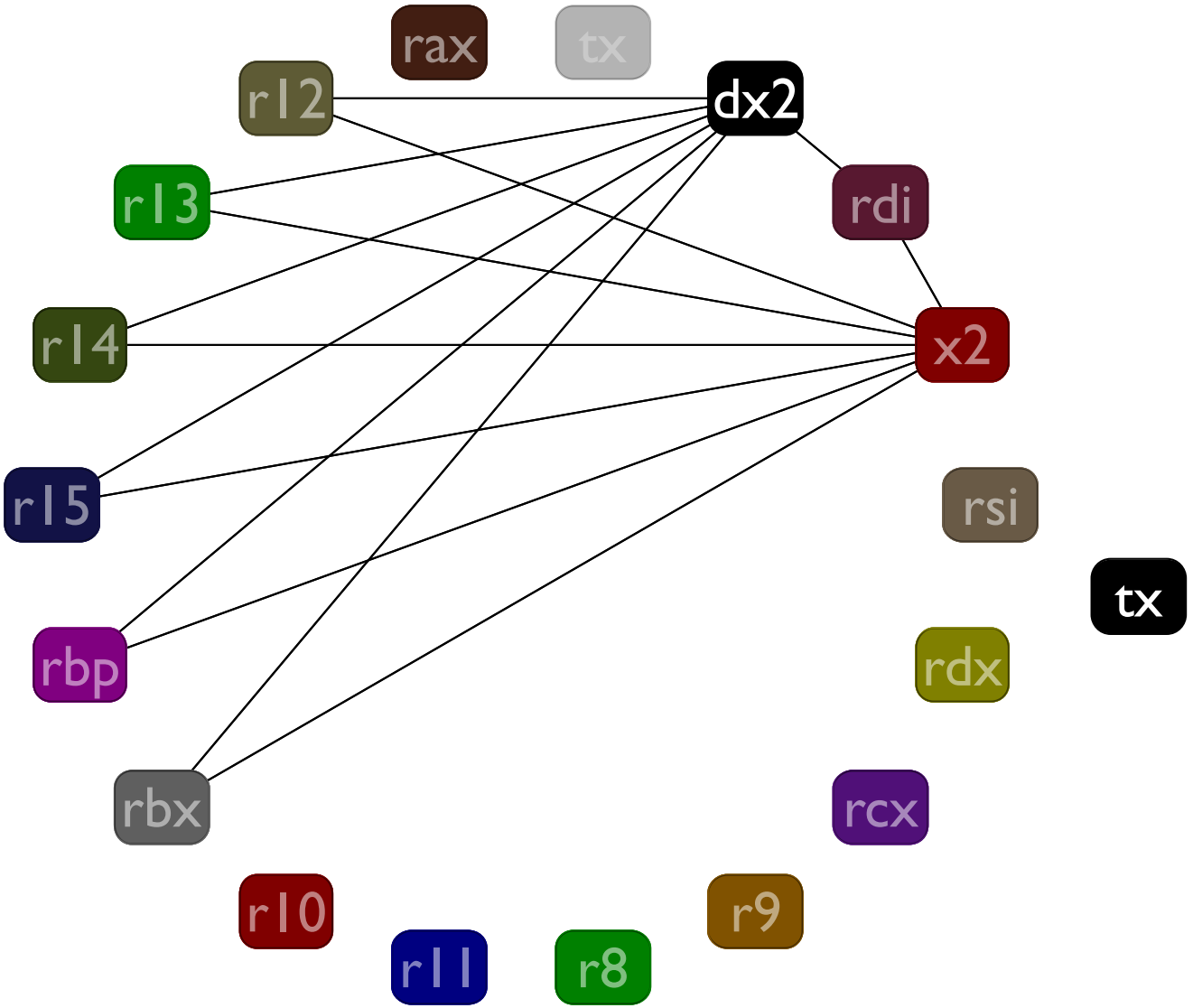

rax
tx
r12
dx2
r13
rdi
r14
x2
r15
rsi
dx2
tx
rbp
rdx
rbx
rcx
r10
r9
r11
r8

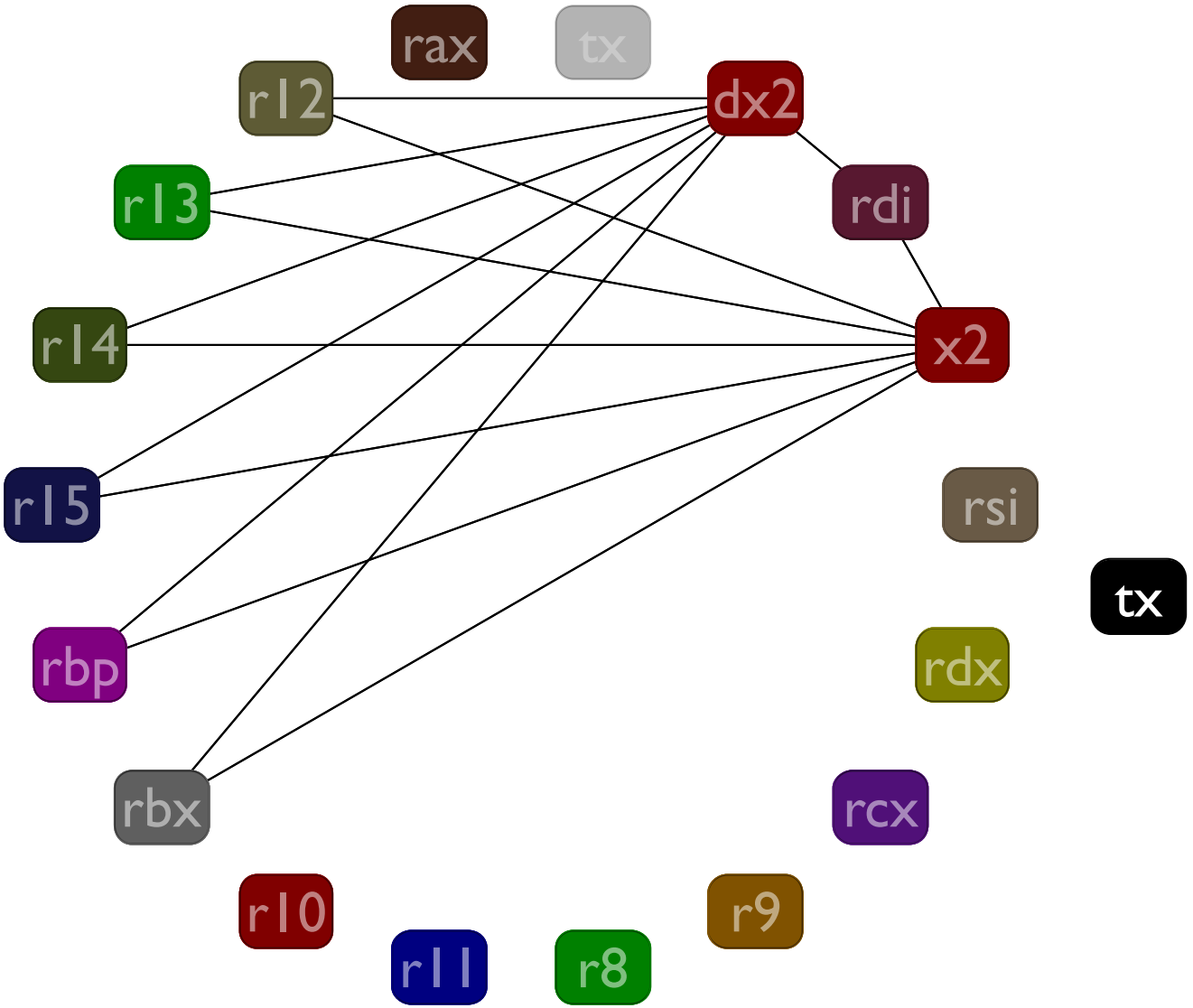

rax
tx
r12
dx2
r13
rdi
r14
x2
x2
r15
rsi
dx2
tx
rbp
rdx
rbx
rcx
r10
r9
r11
r8

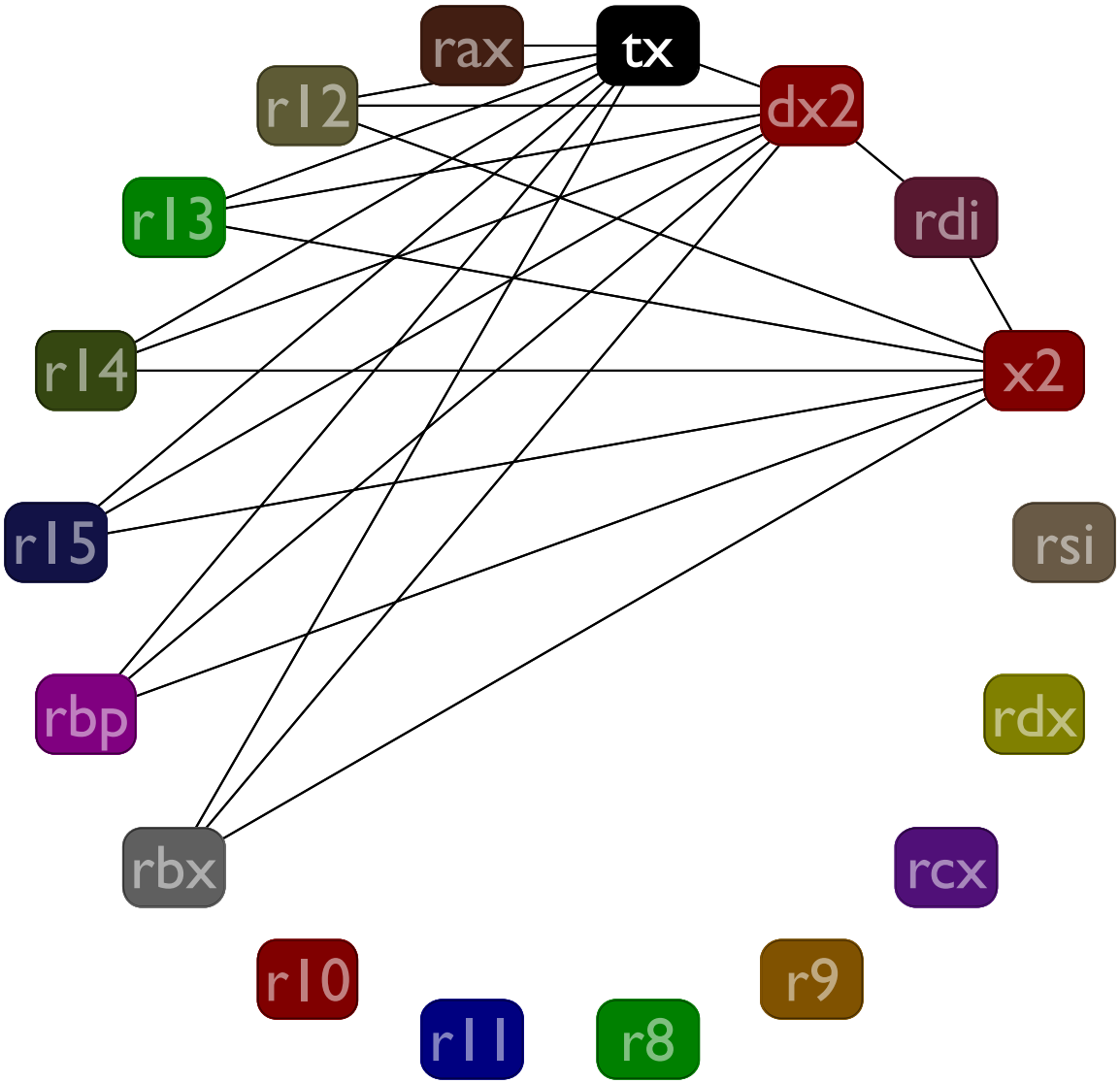

rax
tx
r12
dx2
r13
rdi
r14
x2
x2
r15
rsi
dx2
tx
rbp
rdx
rbx
rcx
r10
r9
r11
r8

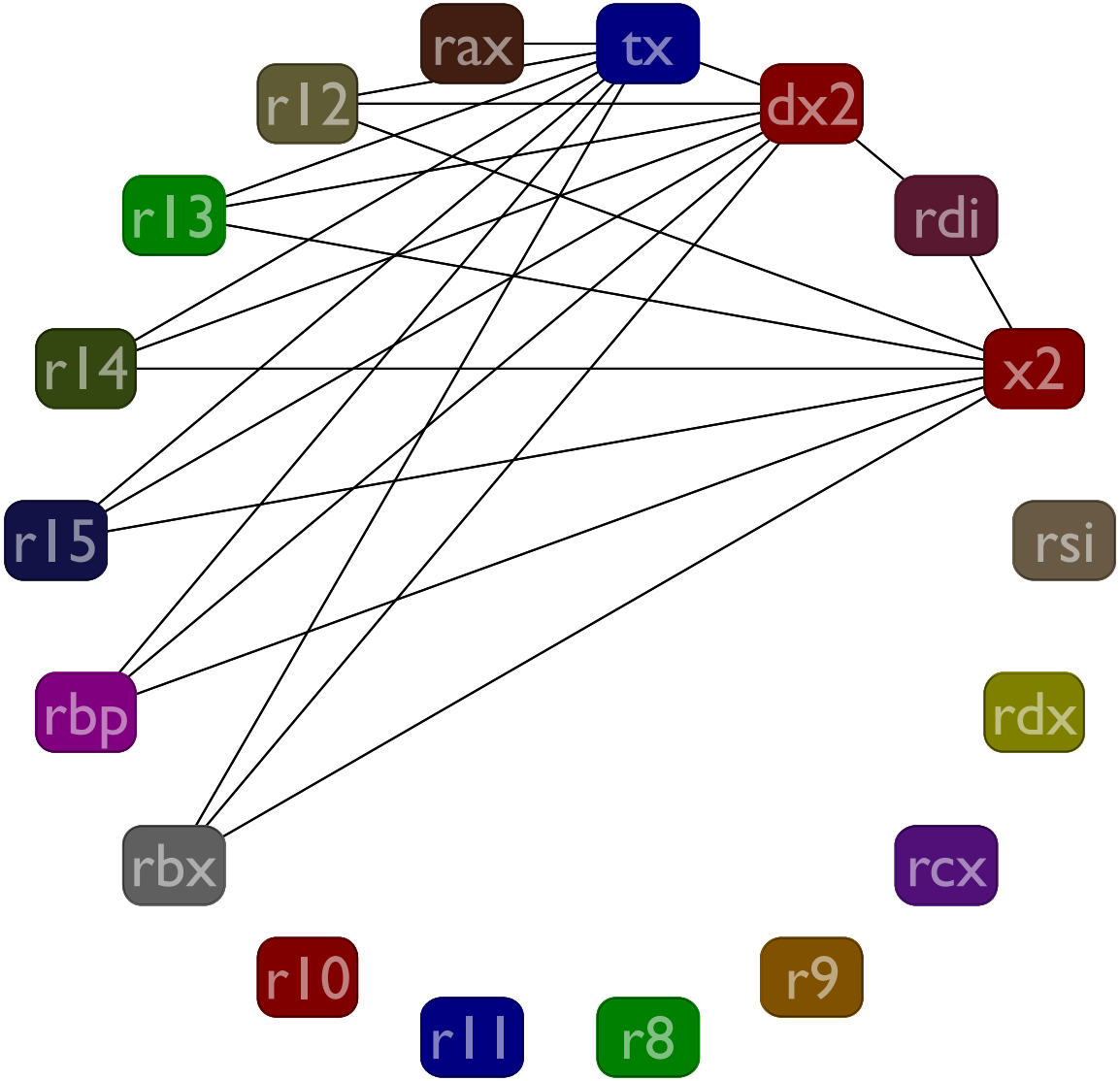

rax
tx
r12
dx2
r13
rdi
r14
x2
r15
rsi
dx2
tx
rbp
rdx
rbx
rcx
r10
r9
r11
r8


rax
tx
r12
dx2
r13
rdi
r14
x2
r15
rsi
dx2
tx
rbp
rdx
rbx
rcx
r10
r9
r11
r8


rax
tx
r12
dx2
r13
rdi
r14
x2
r15
rsi
tx
rbp
rdx
rbx
rcx
r10
r9
r11
r8


rax
tx
r12
dx2
r13
rdi
r14
x2
r15
rsi
tx
rbp
rdx
rbx
rcx
r10
r9
r11
r8


rax
tx
r12
dx2
r13
rdi
r14
x2
r15
rsi
rbp
rdx
rbx
rcx
r10
r9
r11
r8


rax
tx
r12
dx2
r13
rdi
r14
x2
r15
rsi
rbp
rdx
rbx
rcx
r10
r9
r11
r8

Use spilling if you cannot color; make a new program and starts over

.... but this doesn't always work

in that case, just fail; I don't think it is possible for an L5 program to get compiled to an L2 program that will fail to be colorable