Register Allocation

$\boxed{\text{Used}}$

rax r10 r11 rdi
rsp rbp r12

Time to access ✱

register :	1 clock cycle
L1 cache :	~4 clock
L2 cache :	~10
L3 cache :	~40
RAM :	~100 - 300

```
let x = 2 + 4 in        ← []
let y = 3 - 1 in        ← [x]
let z = x + y in        ← [x,y]
let a = after(x) in     ← [x,z]
let b = a - z in        ← [z,a]
before(b)               ← [b]
```

"Liveness"

```
mov rax, 4
mov [rbp-8], rax        r13
mov rax, 8
mov [rbp-16], rax       r14
mov rax, [rbp-8]        r13
add rax, [rbp-16]
mov [rbp-8], rax
mov rax, 6
mov [rbp-16], rax
mov rax, 2
mov [rbp-24], rax       r15
mov rax, [rbp-16]
sub rax, [rbp-24]
mov [rbp-16], rax
mov rax, [rbp-8]
mov [rbp-24], rax
mov rax, [rbp-16]
mov [rbp-32], rax       [rbp-8]
mob rax, [rbp-24]
sub rax, [rbp-32]
mov [rbp-24], rax
mov rax, [rbp-8]
add rax, 2
mov [rbp-32], rax
mov rax, [rbp-32]
mov [rbp-40], rax
mov rax, [rbp-24]
mov [rbp-48], rax
mov rax, [rbp-40]
sub rax, [rbp-48]
mov [rbp-40], rax
mov rax, [rbp-40]
sub rax, 2
```

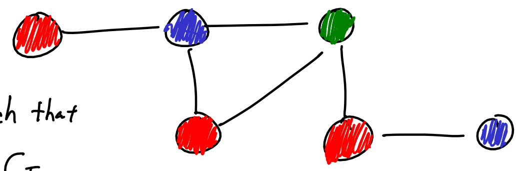
$(\boxed{-16}, \{x \mapsto [rbp-8]\})$

[[rbp-16], [rbp-24], [rbp-32], ...]

[r13, r14, r15, [rbp-8], ...]

① Greedy Algorithm

Graph Coloring



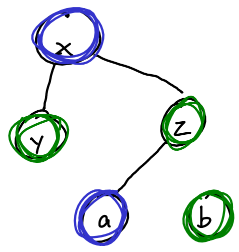
Find a mapping $M : V \mapsto \text{color}$ such that for each edge $\langle V_1, V_2 \rangle$ in a graph G, $M(V_1) \neq M(V_2)$. Minimize codomain of M.

```
let x = 2 + 4 in        ← []
let y = 3 - 1 in        ← [x]
let z = x + y in        ← [x,y]
let a = after(x) in     ← [x,z]
let b = a - z in        ← [z,a]
before(b)               ← [b]
```

Interference Graph

Vertices : conceptual locations
Edges : coexistence
Colors : actual locations

"Coexistence graph"



r13
r14

② Graph Coloring Approximation

⊕ Very good results
⊖ A bit slow

Intermediate Representation (IR) ex. LLVM

```
mov rax, 4
mov loc1, rax
mov rax, 8
mov loc2, rax
mov rax, loc1
add rax, loc2
mov loc1, rax
mov rax, 6
mov loc2, rax
mov rax, 2
mov loc3, rax
mov rax, loc2
sub rax, loc3
mov loc2, rax
mov rax, [rbp-8]
mov loc3, rax
mov rax, [rbp-16]
mov loc4, rax
mob rax, loc3
sub rax, loc4
mov loc3, rax
mov rax, loc1
add rax, 2
mov loc4, rax
mov rax, loc4
mov loc2, rax
mov rax, loc3
mov loc1, rax
mov rax, loc2
sub rax, loc1
mov loc2, rax
mov rax, loc2
sub rax, 2
```

③ Linear Scanning

Basically ①, but when we need new space, we reallocate:

* Move a value in a register to stack
* Put new value in register
* Need old value? Move it back in.

⊖ ~12% worse than ②

⊕ Very fast to generate —— $\overbrace{\text{JIT}}^{\text{just-in-time}}$ compiling