Register Allocations:

- put each variable in a register
- using as few register as possible

B1

```
1  x=0
```

B2

```
2  y=0
```

B3

```
3   t1 = x *24
4   t2 = p + t1
5   t3 = t2 + 8
6   t4 = *t3 (float)
7   t5 = y *24
8   t6 = p + t5
9   t7 = t6 + 8
10  t8 = *t7 (float)
11  t9 = t4 - t8
12  t10 = t9 * t9
13  t11 = x * 24
14  t12 = p + t11
15  t13 = t12 + 16
16  t14 = *t13 (float)
17  t15 = y * 24
18. t16 = p + t15
19. t17 = t16 * 16
20  t18 = *t17
21. t19 = t14 - t13
22. t20 = t19 * t19
23. t21 = t10 + t20
24. t22 = x * 9
25. t23 = t22 * 4
26. t24 = 8 * t23
27. t25 = A + t24
28. t25 = t21
29. y = y + 1
30. if (y ≤ 8) jump 3
```

B4

```
31. x = x + 1
32. if (x = 8) jump 2
```

B5

```
33. return
```

A Basic Block is a maximal ambiguous set of instructions such that the only entry to the block is the first instruction the only exists is the last instruction.

1. First instruction
2. Jump target
3. Instruction after jump (conditional or not)

Optimizations:

local : 1 basic block

globe : All basic block in a procedure

interprocedural : All blocks in a program (anything higer than global)

A variable is said to be "live" if its value can possibly be used later. It is "dead" otherwise.

local liveness analysis

① make assumptions about which variables are live at the end of the block

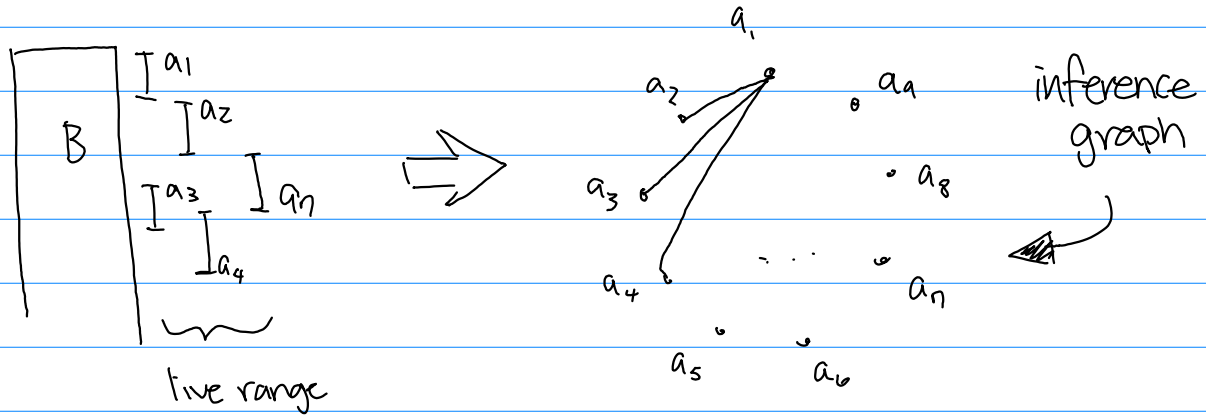
② move up block from bottom, 1 instruction at a time.

a. if instruction $a = b \text{ op } c$

then set b and c to live — gen of bc

set a to dead — kill of a

Live range of variables: interval between gen & kill



Inference Graph: ① draw a node (point) for each variable
② draw an edge a_i to a_j if interval a_i + a_j overlap

Color vertices: Colour theory (no adjacent can be the same)
↳ min. colors = min register

In general, this is NP-hard

local optimizations:

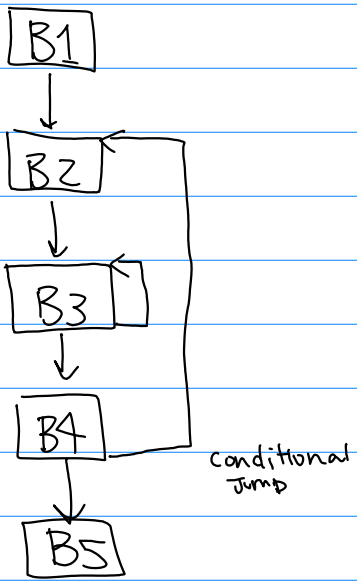
- peephole: use a little window to search for patterns in a code that can be replaced with something better.

```
*t1 = (some_value) t3        *t1 = t3
t2 = *t1  ---------->  t2 = t3
                code improvement
```

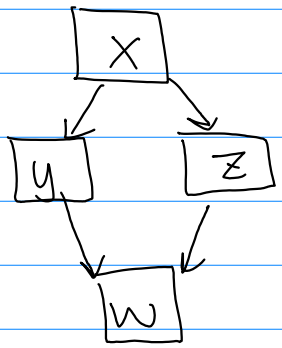
JUMP 14 → JUMP 21

JUMP 21 → JUMP 21

Block Diagram:



B2 dominates B3, therefore B2 dominates B4.



x dominates everything, but y does not dominate w because it can go through z.

Algebraic Simplifications

$0 * x = x * 0 \rightarrow 0$

$1 * x = x * 1 \rightarrow x$

$0 + x = x + 0 \rightarrow x$

$x - 0 \rightarrow x$

$0 - x \rightarrow$ negate? $\rightarrow$ depending on processor

$(float)\, x / 2 \rightarrow x * 0.5 \rightarrow$ multiply usually faster than divide

$x * 2 \rightarrow x + x$

$x << 1$

$pow(x, 2) = x^2 \rightarrow x * x$

$x^2 > * > + >$ shift

Constant folding

Compile-time replacement of computation by value

$t15 = 16 + 29$
$t15 = 45$: Constant folding because generally look at constant variables and dont give side effects

← pure function

$t16 = factorial(7)$
$t16 = 5040;$

- $t_1 \leq t_2$: this never overflow
- $t_2 - t_1$ if $t_3 \leq 0$: but this one does