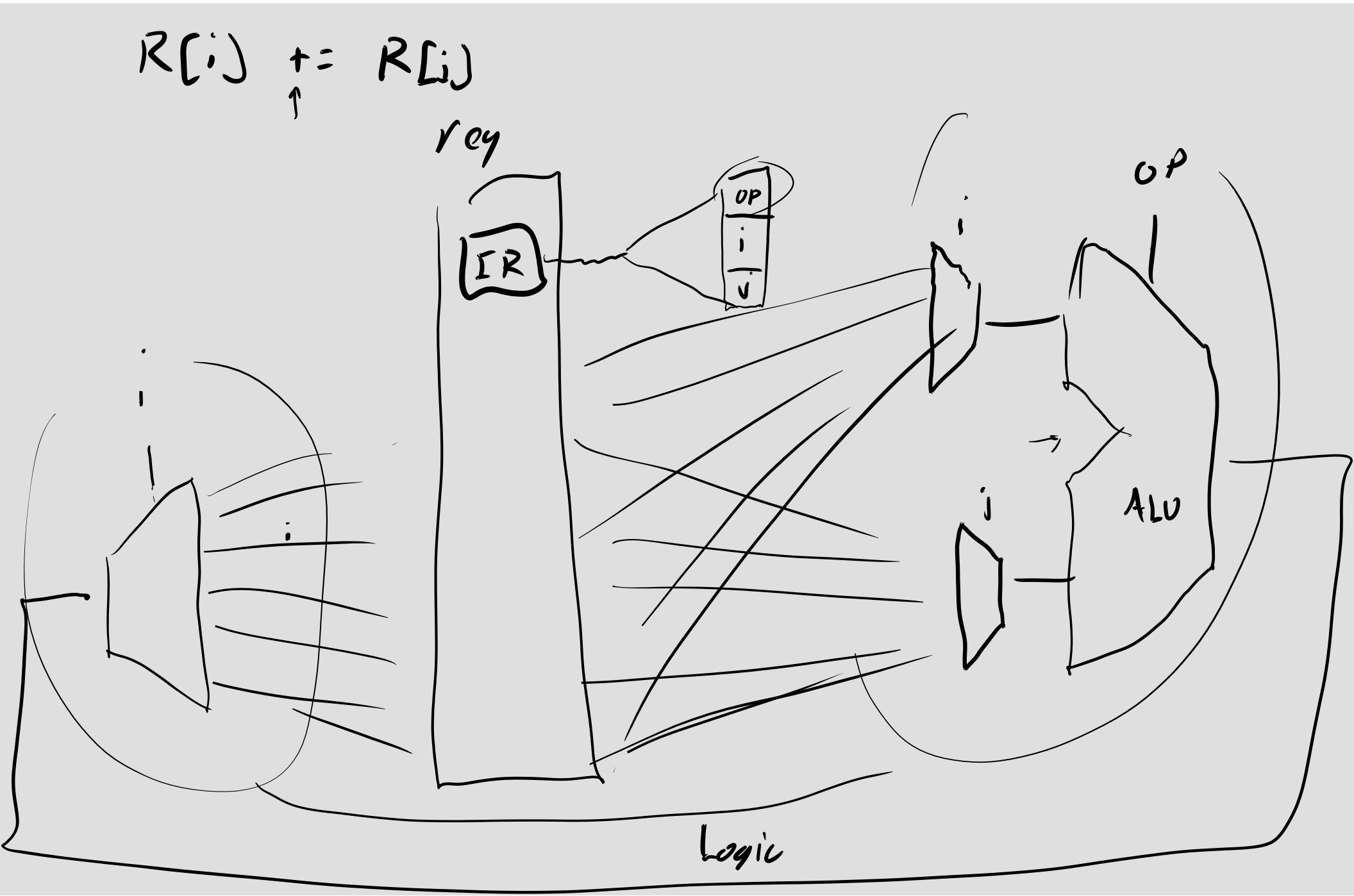

Beware
the
COA
Constrictor

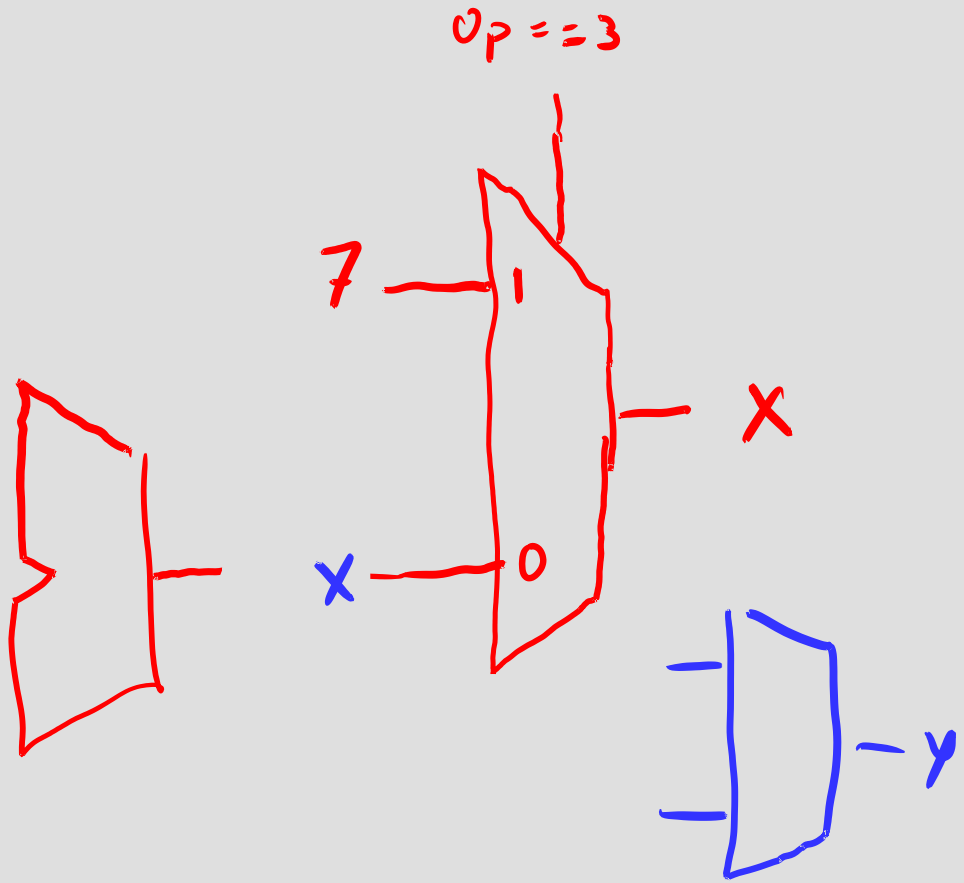

R[i] += R[j]
reg
IR
OP
i
i
OP
i
i
ALU
Logic

RTL

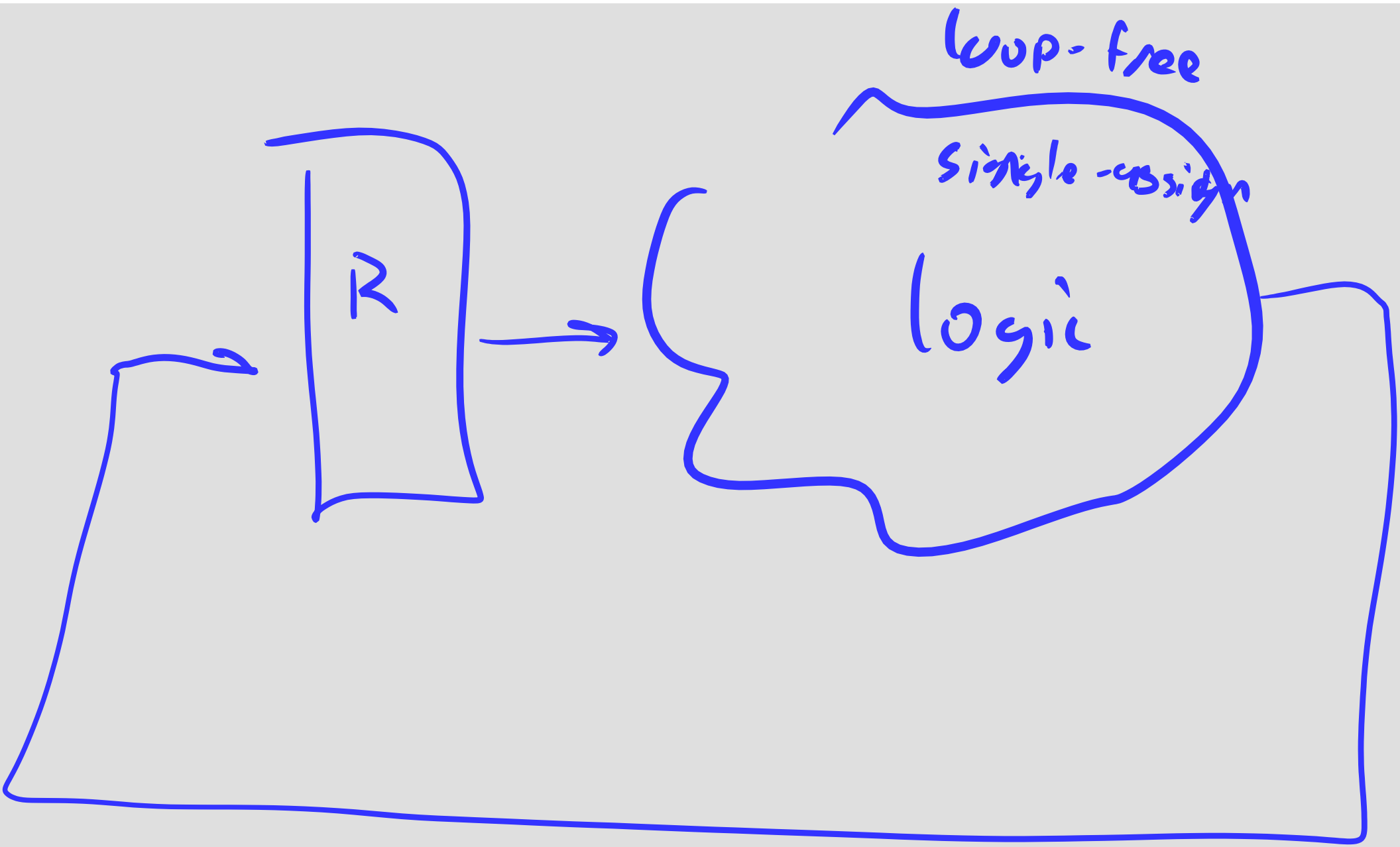
wire
var

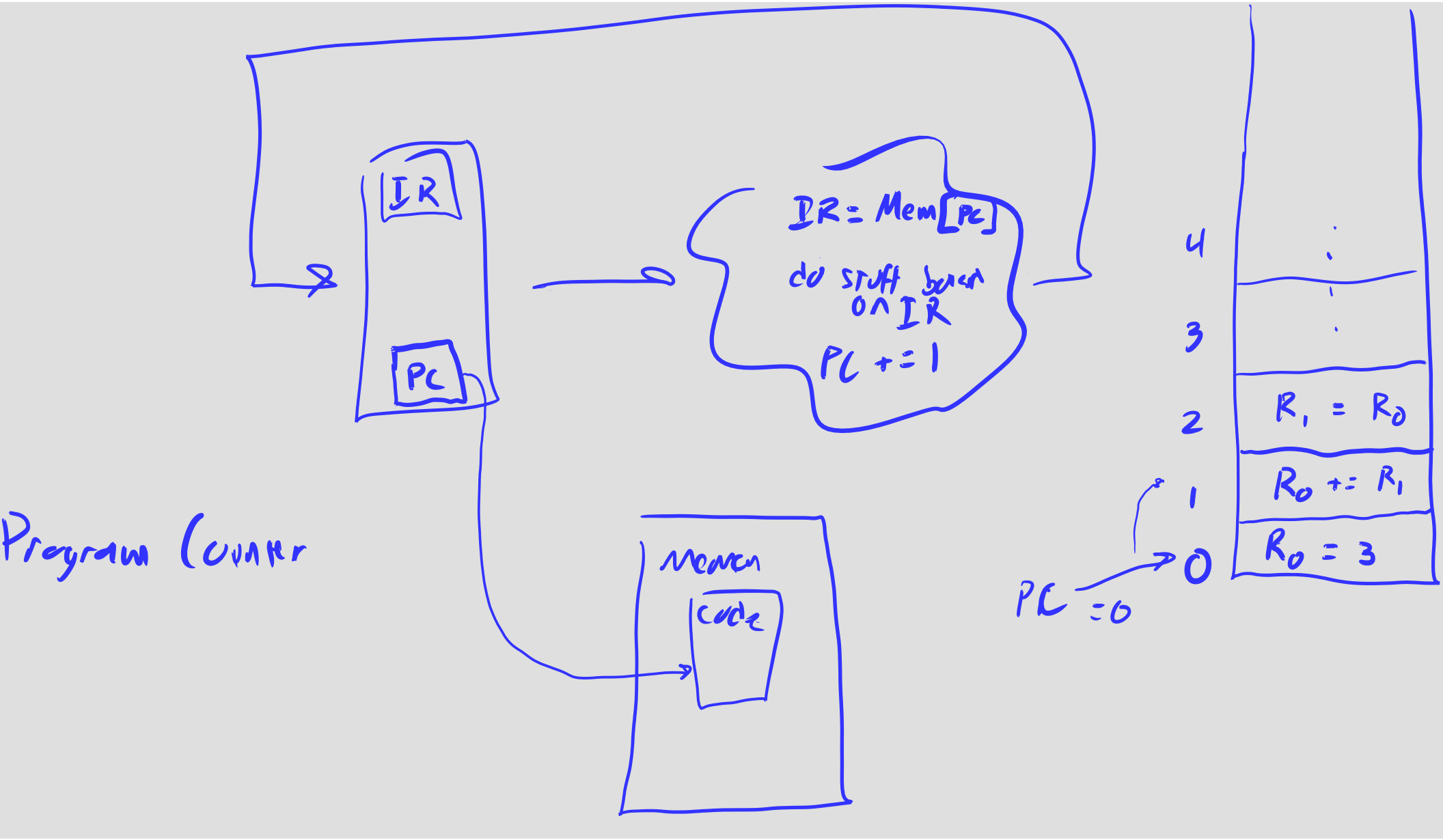
assy once Per cycle

expressions, no loops or ...




```
if (op == 3) {
    x = 7
} else {
    op, a, b → ALU
    y = ALU_result
}
```


loop-free
R
logic


IR
PC
IR = Mem[PC]
do stuff based on IR
PC += 1
Memory
code
Program Counter
4
3
2
1
0
R1 = R0
R0 += R1
R0 = 3
PC = 0

Programmable Computer

- have code in memory
  - ↳ instructions, actions Encoded as bits
- Registers + logic do work
- sending btwn Reg + Mem

Von Neumann Model

PC

IR

Reg Bank

Memory

Peripherals I/O

Mem[PC] → IR

IR ⟨ what to do ⟩ what to do it to

logic + muxes to do it

~~PC += size of IR~~

PC = new based on PC + Instruction

address

0 x = 1

2 y += x

3 y += x

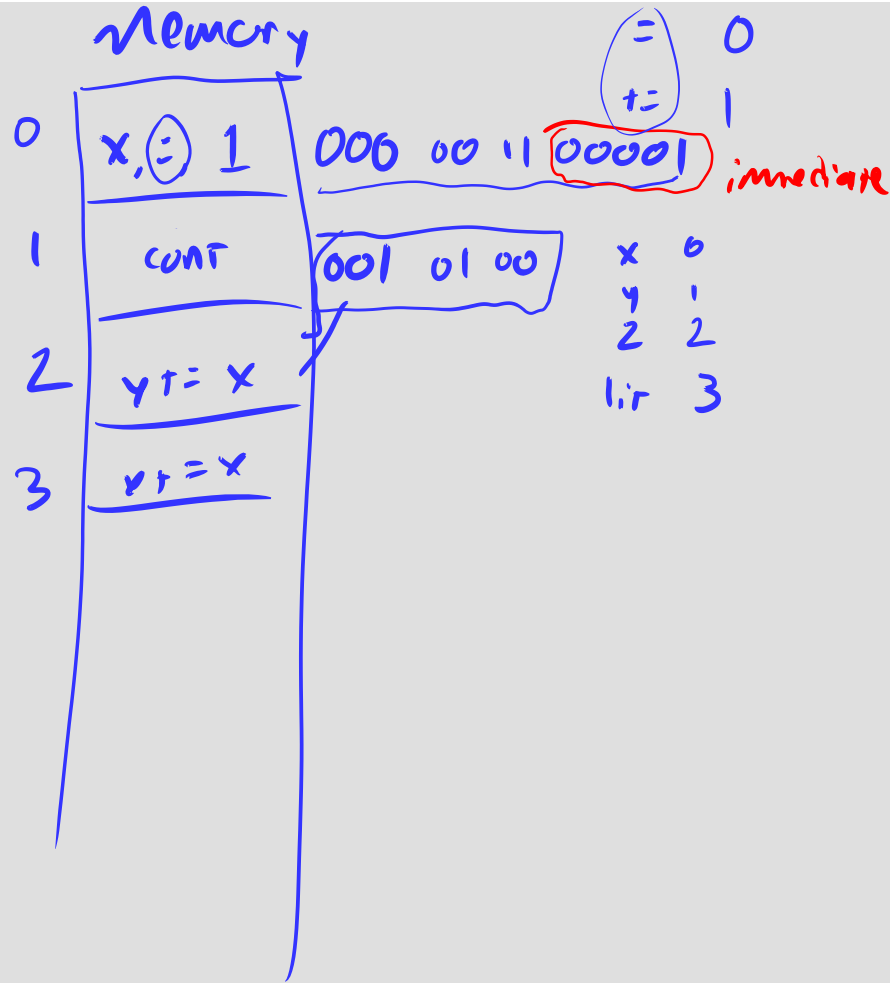
next

next

next

address = index into memory

OP	PC
= lit	+= 2
+= reg	+= 1
+= reg	+= 1



while (true) {

x += x - 1

tmp = x - 1

x += tmp

}

tmp = x

tmp -= 1

0 next

1 next

go to addr 0 next 2

= src, dest

+= src, dest

-= src, dest

go to address

any

PC

0

1

2

0

0	y = x
1	y -= 1
2	x += y
3	goto 0

Jump

Conditional Jump

Jump to <addr> if <reg> is <op> 0

<op>: < <= > >= == !=