Pointer Codegen Example
code generation (assembly)

- Code generation in assembly is easy: keep the same CodeObject, but switch whether temporary is an l-val or an r-val
• \(* x = * (y + 7)\)
code generation (assembly)

• \( \star x = \star (y + 7) \)
**code generation (assembly)**

- \( * x = \ast (y + 7) \)

```
L/R: L
Var: x

L/R: L
Tmp: t1
Code:
lw t1, 0x4004

L/R: L
Var: y

Assignment

BinOp (+)

de-ref

Id (x)

Id (y)

Lit (7)

L/R: R
Val: 7
```
* x = *(y + 7)
code generation (assembly)

* \( x = \ast (y + 7) \)
**code generation (assembly)**

- \(* x = *(y + 7)\)
code generation (IR)

• Code generation for IR is similar

• Track whether IR temporary holds an l-value or an r-value (e.g., use ‘$’ as prefix for r-value, ‘@’ as prefix for l-value)

• Introduce two new IR nodes:
  • **ADDROF a, b**: store the address of operand b in a (if b is a variable, a holds the variable address; if b is an l-value temporary, a is the temporary, just as an r-value)
  • **DEREF a, b**: store the value of operand b in a as an address (if b is a variable or an l-value temporary, load from b and store the result in a as an l-value; if b is an r-value temporary, a is the temporary, just as an l-value)
register allocation

• Now that we have pointers, we have aliasing!

• Simple solution: treat all locals/globals as aliased to each other: cannot stay in registers. Write back on every store, free after every load

• Slightly more complicated: only variables that have ever had an ADDROF operation applied to them can be aliased

• More complex: perform **pointer analysis** (stay tuned!)
next: memory allocation