Generating Code

what kind of code to generate?

- ASTs are not executable code
- Walk over AST to generate assembly instructions
  - Compiler project: RiscV assembly
  - Typically: generate assembly assuming unlimited (virtual) registers, make the code work with fewer registers later

from ASTs to code

- To generate code, we can perform a post-order *walk* of the AST
  - Walk over AST, for each sub-tree, generate code *for that subtree*, combine code from multiple subtrees to generate code for larger tree:

```
CodeObject generate_code() {
  //pre-processing code
  CodeObject lcode = left.generate_code();
  CodeObject rcode = right.generate_code();
  return generate_self(lcode, rcode);
}
```

what is a codeobject?

- Keeps track of information for segments of code associated with an AST subtree
    - List of instructions that correspond to the code for that subtree
    - Register where result of expression is stored (if codeobject is for an expression)
    - Whether code object holds code or other information (constant, variable name)
    - Whether register stores an **l-value** or an **r-value**

l-values vs r-values

- L-values: addresses which can be stored to or loaded from
- R-values: data (often loaded from addresses)
- Expressions operate on R-values
  - Assignment statements: L-value := R-value
- Consider the statement a := a + 1
- the a on LHS refers to the memory location referred to by a and we store to that location
- the a on RHS refers to data stored in memory location referred to by a so we will load from that location to produce the R-value

simple cases

- Generating code for constants/literals
  - Simple option: store constant in register (using load immediate instruction)
  - More complicated: defer generating code, pass constant up in codeobject with constant flag (lets you use other immediate instructions later)
- Generating code for identifiers
  - Is this an address? Or data? Depends on whether it's on the LHS or the RHS!
    - If on LHS, need to keep it as memory location to store to
    - If on RHS, need to load from it
  - Simple solution:
    - Pass identifier up to next level, wait until we see how it is used to generate code
  - Mark it as an L-value (it's not yet data!)

generating code for expressions

- Allocate a fresh virtual register for result of expression
- Examine codeobjects from subtrees
  - If result registers are L-values, load data from them into new registers (need to operate on data, not addresses)
    - Generate code to perform operation
  - If code object flagged constant, can perform operation immediately
    - No need to perform code generation!
- Store result in freshly-allocated virtual registers (temporaries)
  - Is this an L-value or an R-value?
  - Return code for entire expression

generating code for assignments

- Store value of temporary from RHS into address specified by temporary from LHS
- Why does this work?
  - Because temporary for LHS holds an address
  - If LHS is an identifier, we passed the identifier up itself as an L-value (get actual address from symbol table)
  - If LHS is complex expression

```
int *p = &x

*(p + 1) = 7;
```

it still holds an address, even though the address was computed by an expression next: example