Compilers

Arthur Hoskey, Ph.D.
Farmingdale State College
Computer Systems Department

Today's Lecture

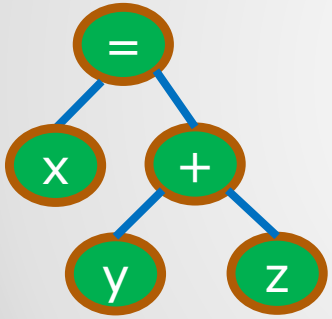
- Code Generation

Running Code

- A computer's CPU runs all code.
- Each CPU has its own specific instruction set that it knows how to run.
- A program in some source language must be converted into a form that can be executed by a specific computer.
- Here are some compiler output types:
  - Machine code for a specific CPU
  - Assembly language code
  - Java bytecode
- For example, the Java Virtual Machine (JVM) converts Java bytecode into machine code (Java Bytecode → Machine Code).

- Code will be generated from an abstract syntax tree.
- We will be generating pseudo-assembly code.

Abstract Syntax Tree
(Intermediate Representation



Assembly Code

```
var x
var y
var z
load r1, y
load r2, z
add r1, r2, r3
store r3, x
```

The AST is converted to assembly language code

Code Generation

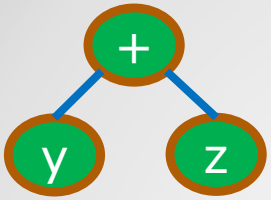
- Code will be generated by traversing the abstract syntax tree.
- Use a depth first traversal to generate code (a breadth first traversal would not work).

Code Generation

- We will first look at generating Java code from an abstract syntax tree…

Generating Java Code from an AST

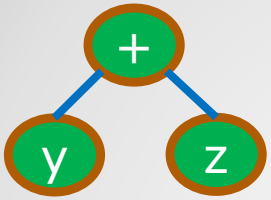
- Code generation for expression.
- What is the Java code?



Code Generation – Expressions

Code Generation - Expression

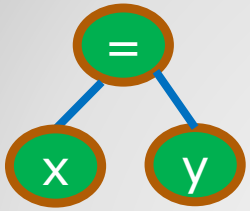
- Code generation for expression.
- What is the Java code?



- Java code

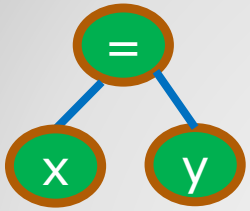
y + z

- Code generation for assignment.
- What is the Java code?



Java – Assignment

- Code generation for assignment.
- What is the Java code?



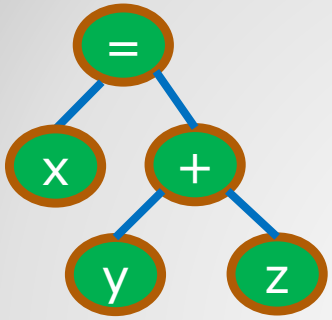
- Java code

```
x = y;
```

A semicolon needs to be added at the end for Java.

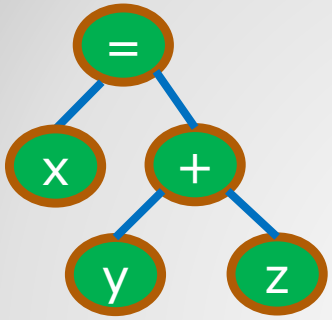
Java – Assignment

- Code generation for assignment with expression.
- What is the Java code?



Java – Assignment with Expression

- Code generation for assignment.
- What is the Java code?

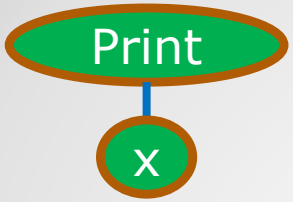


- Java code

```
x = y + z;
```

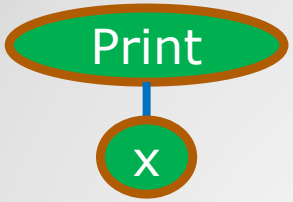
Java – Assignment with Expression

- Code generation for assignment.
- What is the Java code?



Java – Print

- Code generation for assignment.
- What is the Java code?



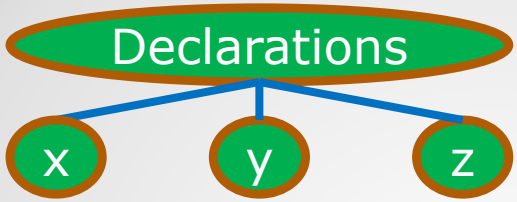
- Java code

```
System.out.println(x);
```

Java uses System.out.println to print to the console

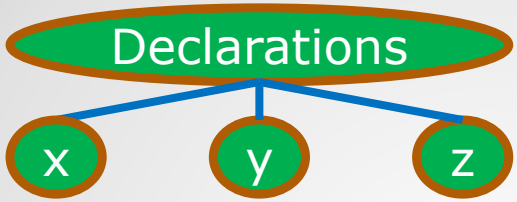
Java – Print

- Code generation for declarations.
- What is the Java code? Assume that only the int data type allowed.



Java – Declarations

- Code generation for declarations.
- What is the Java code? Assume only int data type allowed.



- Java code

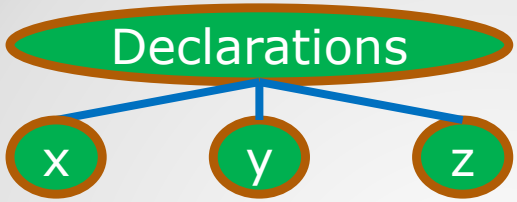
```
int x;
int y;
int z;
```

Used three separate statements for the declarations

- What would be another way to do this in Java?

Java – Declarations

- Code generation for declarations.
- What is the Java code? Assume only int data type allowed.



- Java code

```
int x;
int y;
int z;
```

Used three separate statements for the declarations

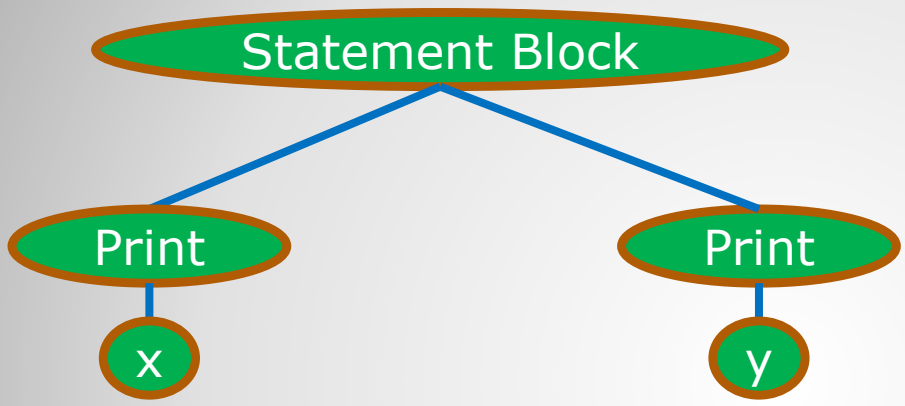
- What would be another way to do this in Java?

```
int x, y, z;
```

All three declared on same line

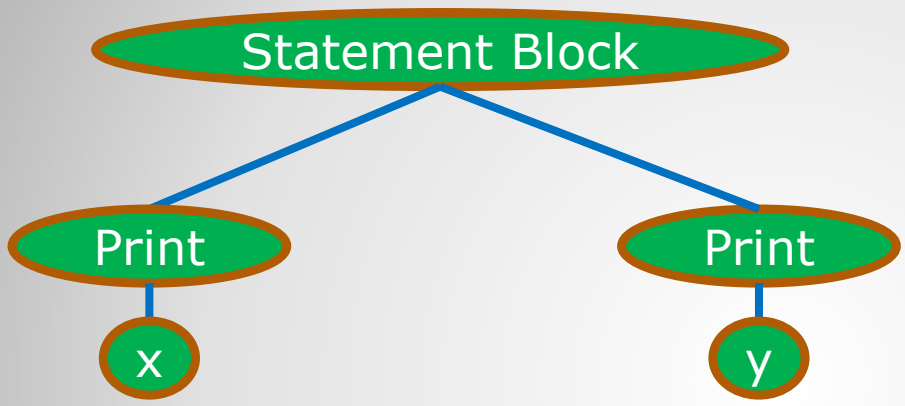
Java – Declarations

- Code generation for statement block.
- What is the Java code? What is Java syntax for a block?



Java – Statement Block

- Code generation for statement block.
- What is the Java code? What is Java syntax for a block?



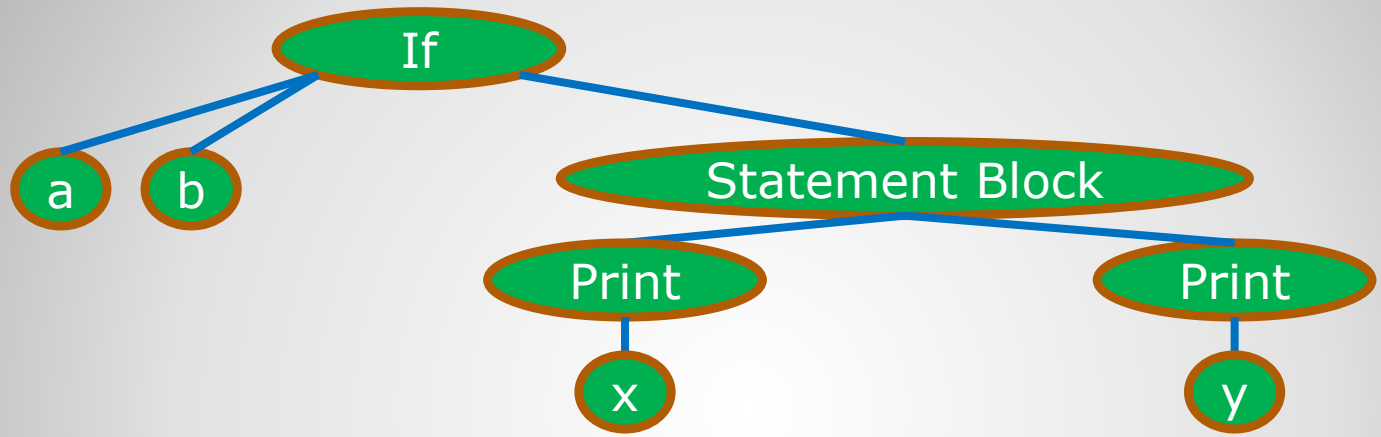
- Java code

```
{
   System.out.println(x);
   System.out.println(y);
}
```

Curly braces were added before and after the statements

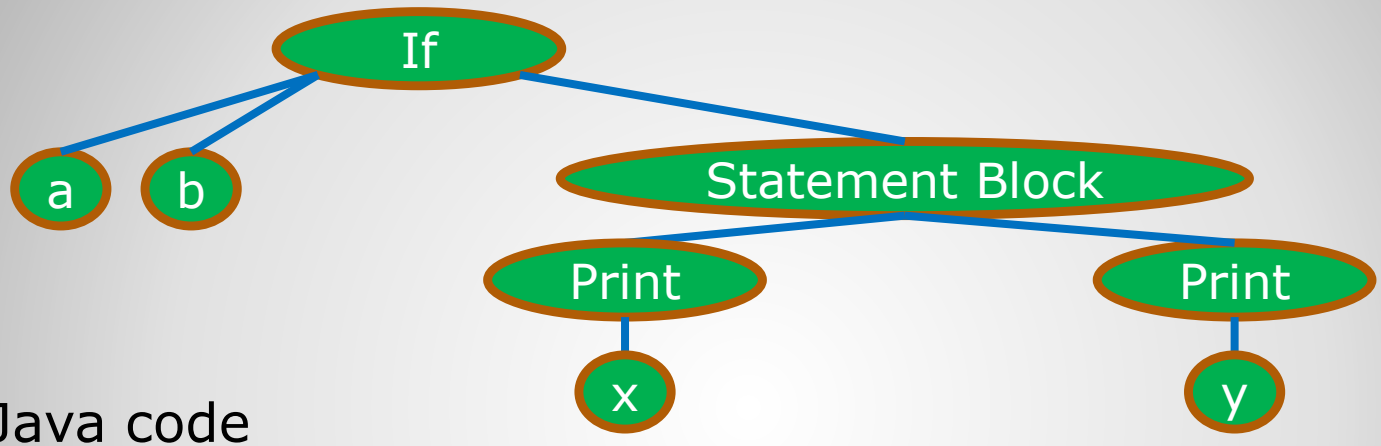
Java – Statement Block

- Code generation for if. Use equality test on a and b.
- What is the Java code?



Java – If

- Code generation for if. Use equality test on a and b.
- What is the Java code?



- Java code

```
if (a == b)
{
   System.out.println(x);
   System.out.println(y);
}
```

If test

Start statement block

End statement block

The if test is followed by the statement block

Java – If

- Now we will look at generating assembler code from an abstract syntax tree…

Generating Assembler Code from an AST

Assembler vs Java

- Generating assembler code is harder than Java (or any other high-level language).
- In general, multiple assembler instructions are required to do what one high-level instruction does.
- The following line of Java code assigns the value y to x:

```
x = y;
```

- The following code does the same in assembler:

```
load r1, y
store r1, x
```

The y variable must first be loaded into a register before it can be copied into x

Put the value into x from register r1

- Two assembler instructions are required to do the assignment.

Assembler vs Java

- Here is an example of doing an addition then assigning the result to another variable.

- Java code:

```
x = y + z;
```

- The following code does the same in assembler:

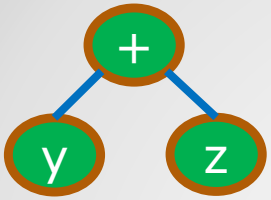
```
load r1, y
load r2, z
add r1, r2, r3
store r3, x
```

Load the y and z variables into registers in preparation for running an add instruction

Add r1 (contains y) and r2 (contains z) and put result of add in r3

Put the result of the add into variable x

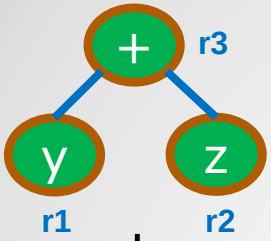
- Code generation for expression.
- What is the assembler code?



Assembler - Expression

Assembler - Expression

- Code generation for expression.
- What is the assembler code?



- Assembler code

```
load r1, y
load r2, z
add r1, r2, r3
```

The result of the addition is being store in register r3

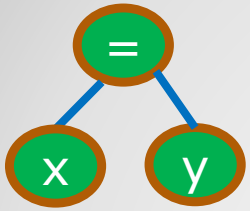
There needs to be a way to choose which register the add will write the result to. If r3 were being used by another part of the program, using it here would create an error. This problem is related to the subject of register allocation.

Register Allocation

- **Register Allocation** – How do we assign variables and temporary values to registers.
- A CPU has a finite number of registers so we cannot give every variable and every temporary value exclusive use of a register.
- We need to keep track of which registers are currently used and unused.
- Values for subexpressions should be stored in a register (if possible).
- If all registers are in use, we can store a value in a variable instead (this is called "spilling" the registers).

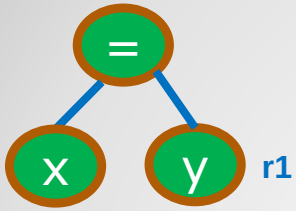
Assembler – Assignment

- Code generation for assignment.
- What is the assembler code? Which registers are used?



Assembler – Assignment

- Code generation for assignment.
- What is the assembler code? Which registers are used?

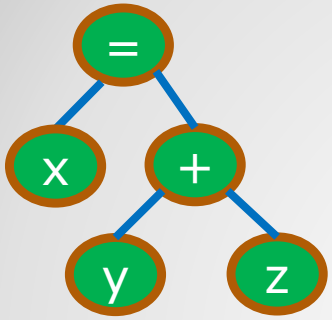


- Assembler code

```
load r1, y
store r1, x
```

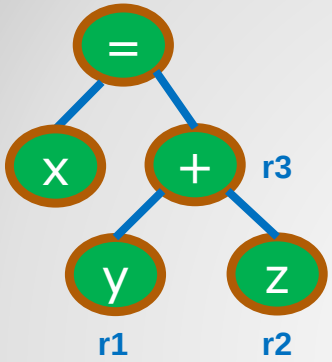
Y needs to be loaded into a register first then it can be stored into x from there

- Code generation for assignment.
- What is the assembler code? Which registers are used?



Assembler – Assignment with Expression

- Code generation for assignment.
- What is the assembler code? Which registers are used?

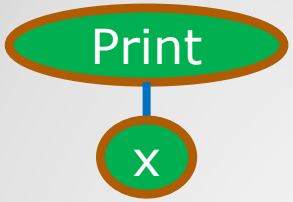


- Assembler code

```
load r1, y
load r2, z
add r1, r2, r3
store r3, x
```

Assembler – Assignment with Expression

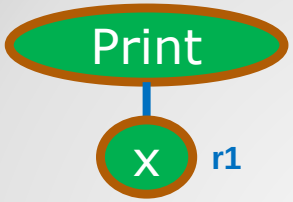
- Code generation for assignment.
- What is the assembler code? Which registers are used?



Assembler – Print

Assembler – Print

- Code generation for assignment.
- What is the assembler code? Which registers are used?



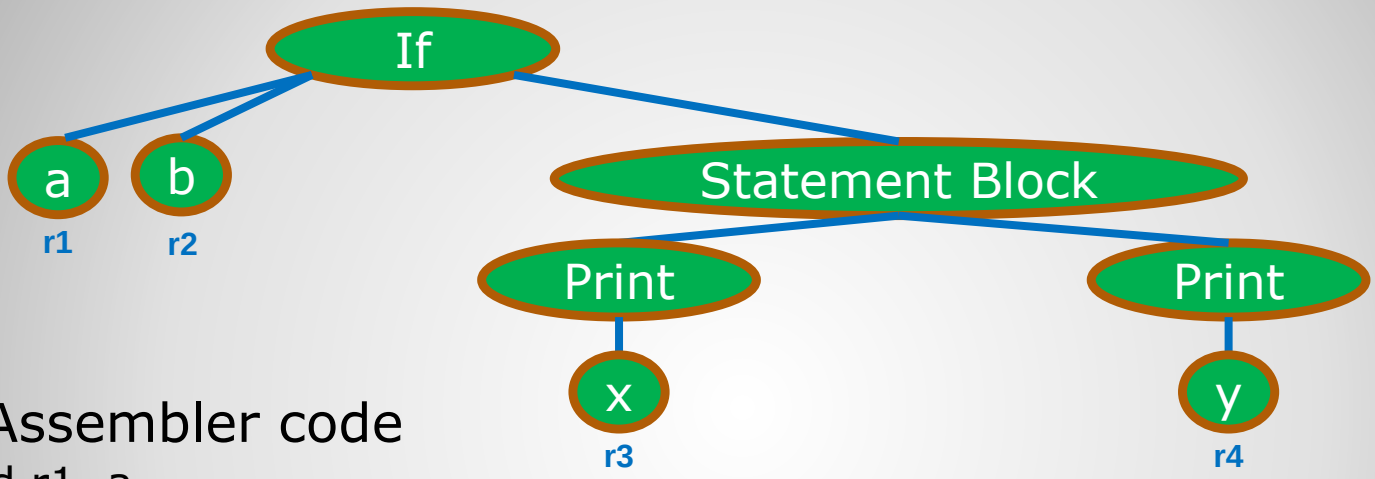
- Assembler code

load r1, x
print r1

Note: Actual assembler code for a print method will require more code for a method call. We are using a very simplified assembly language.

Assembler – If

- Code generation for if. a and b are used in the test.
- What is the assembler code? Which registers are used?



- Assembler code

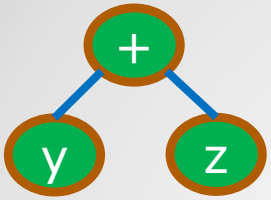
```
load r1, a
load r2, b
branchNotEqual r1, r2, labelEndIf
load r3, x
print r3
load r4, y
print r4
: labelEndIf
```

Branch not equal is being used (instead of branch equal) because it should skip the statements if the values in the test are not equal. If r1 and r2 are not equal, it jumps to labelEndIf.

- Code generation during traversal…

Code Generation During Traversal

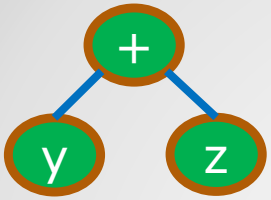
- What are the preorder, inorder, and postorder traversals of the following AST?



Depth First Traversals of AST

Depth First Traversals of AST

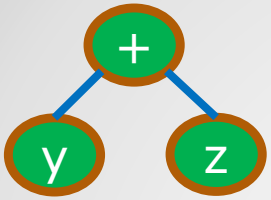
- What are the preorder, inorder, and postorder traversals of the following AST?



- Preorder: + y z
- Inorder: y + z
- Postorder: y z +

Which is best to use when traversing to generate code?

- What are the preorder, inorder, and postorder traversals of the following AST?



- Preorder: + y z
- Inorder: y + z
- **Postorder: y z +**

Which is best to use when traversing to generate code?

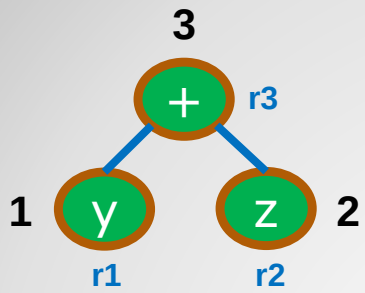
ANSWER: POSTORDER

- **In general, we need to get the operands first before we can generate code for an operator.**
- **In the above example, we need to get the y and z values for the add operation.**

Depth First Traversals of AST

Code Generation During Traversal

- Use a postorder traversal to generate code.



- 1 – Visit node y. This will generate code to put y in a register.

load r1, y

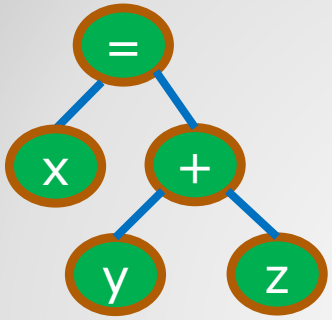
- 2 – Visit node z. This will generate code to put z in a register.

load r2, z

- 3 – Visit node +. This will generate code to add y (in r1) and z (in r2). The result will be put in an open register (r3).

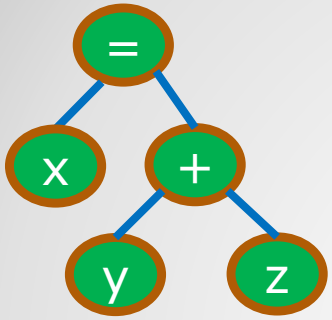
add r1, r2, r3

- What is the postorder traversal?



Assembler – Assignment with Expression

- What is the postorder traversal?



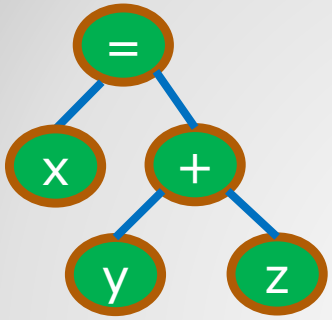
What code will be generated for node x?

- Postorder: x y z + =

Assembler – Assignment with Expression

Assembler – Assignment with Expression

- What is the postorder traversal?

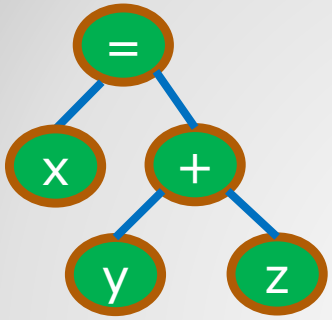


- Postorder: x y z + =

What code will be generated for node x?

ANSWER: NOTHING. THE ASSIGNMENT (=) DOES NOT HAVE TO CALL A METHOD TO VISIT THAT NODE. IT CAN JUST GET THE VARIABLE NAME FROM NODE X. IT WILL STORE THE RESULT IN THIS VARIABLE NAME WHEN GENERATING CODE FOR THE = NODE.

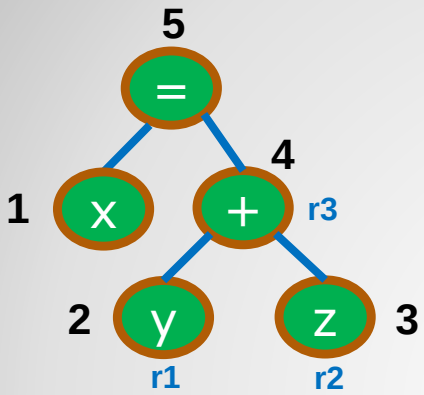
- Describe what happens in a postorder traversal to generate code for this AST. What code is generated at each step?



Assembler – Assignment with Expression

Assembler – Assignment with Expression

- Describe what happens in a postorder traversal to generate code for this AST. What code is generated at each step?



- 1 – No need to visit for assignment.
- 2 - Visit node y. This will generate code to put y in a register.

load r1, y

- 3 - Visit node z. This will generate code to put z in a register.

load r2, z

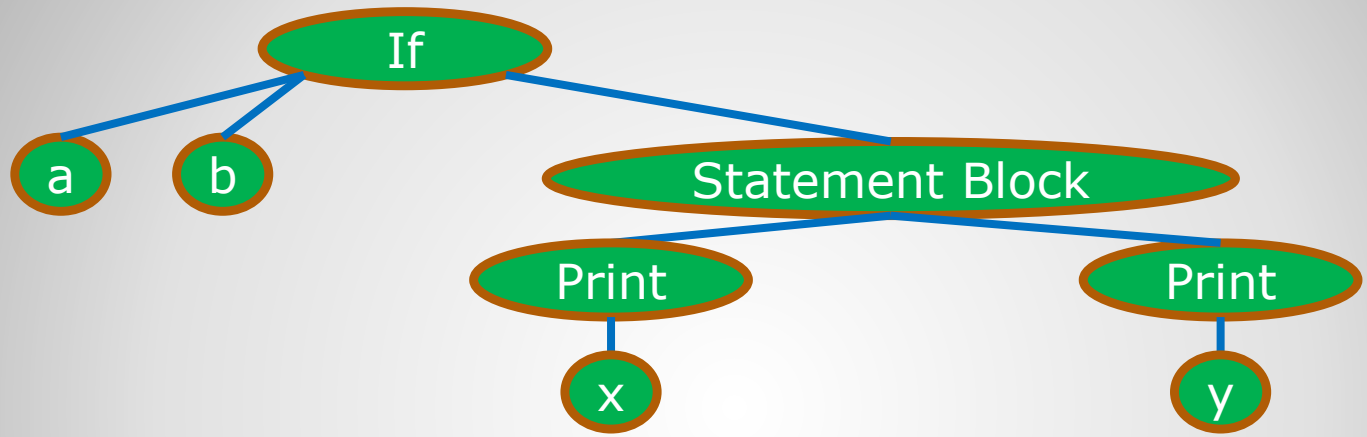
- 4 - Visit node +. Generate code to add y (in r1) and z (in r2). Put result in r3.

add r1, r2, r3

- 5 – Visit node =. Generate code to store int x from r3.

store r3, x

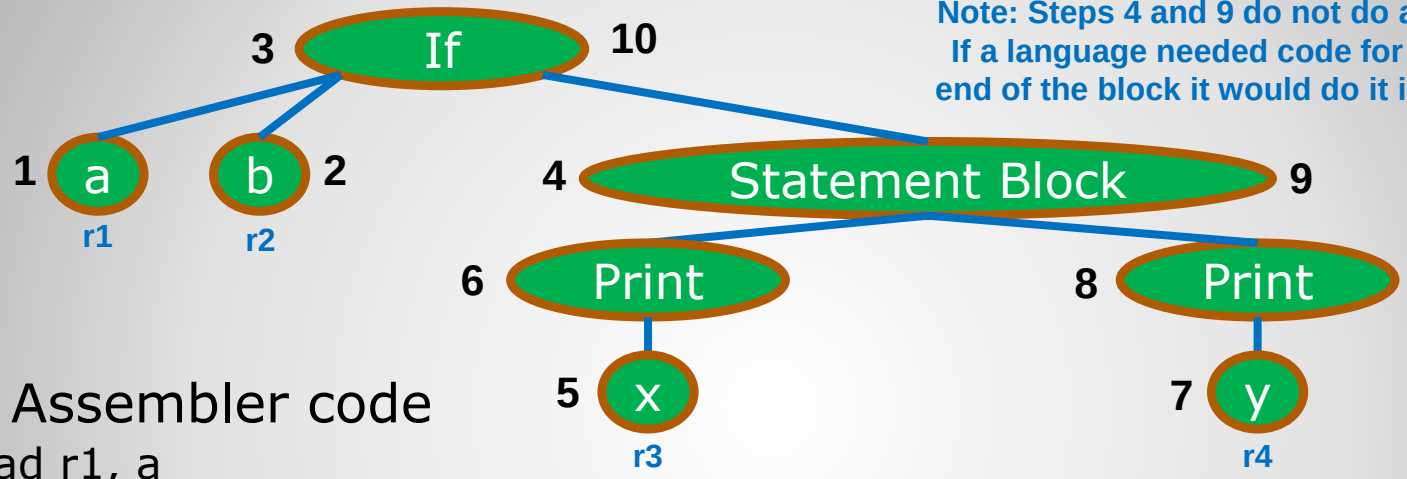
- Describe what happens in a traversal to generate code for this AST. Note: This will NOT be a strict postorder traversal.



Assembler – If

Assembler – If

- Describe what happens in a traversal to generate code for this AST. Note: This will not be a strict postorder traversal.



Note: Steps 4 and 9 do not do anything here. If a language needed code for the start and end of the block it would do it in these steps.

- Assembler code

```
load r1, a
load r2, b
branchNotEqual r1, r2, labelEndIf
load r3, x
print r3
load r4, y
print r4
: labelEndIf
```

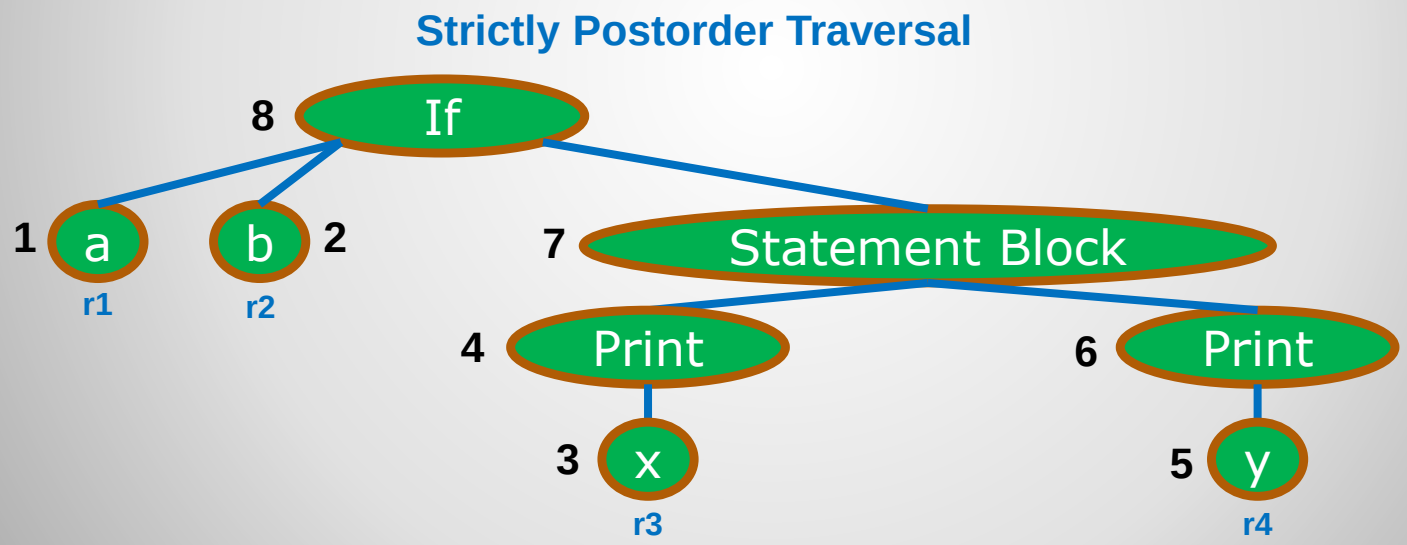
After visiting nodes a and b it writes the branchNotEqual statement. It will then visit the statement block.

Statement Block Code

Branch not equal is being used (instead of branch equal) because it should skip the statements if not equal (jumps to labelEndIf).

Write label after processing statement block

- We could perform a strictly postorder traversal if we wanted.
- Basically, the if node should gather all information before generating any code.
- Nodes a and be should return the registers they are stored in.
- The statement block should gather all code in its block and return the whole code block to the if node.
- The if node would then generate all its code from the gathered information from nodes a, b, and statement block.



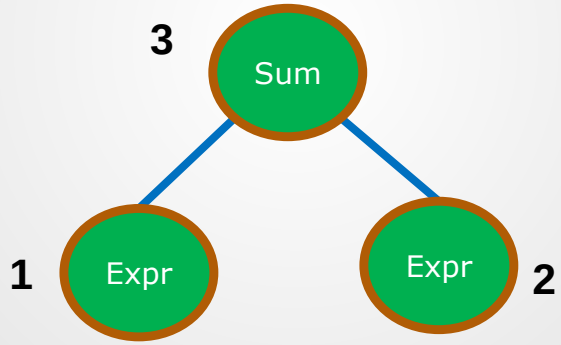
If – Strictly Postorder Traversal

- Pseudocode for code generation…

Pseudocode for Code Generation

- Implement a code() method for every AST node.
- Calls to the code() method should do the following:
  - Generate assembly code for the node. This code should be added to the end of a List<String> that contains all the assembly code (each item is one line of code).
  - Return a register id. This register id can then be used by the parent node to help the parent node generate its code (can return "" for the register id if it does not need one).

Sum's code() will call code() on its left and right children. It will use the register ids returned by each child call when generating its code for the add instruction.



Left child call to code() will generate a load instruction and return the register id

Right child call to code() will generate a load instruction and return the register id

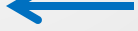
Code Generation Flow

Next Open Register Helper Method

- Some functions need an open register for the result of their operations.
- We will use a trivial register allocation method to make things easier.
- Create a variable to hold the value of the next open register.
- Each time a register is needed, use that value and then increment it.
- A non-trivial algorithm would reuse the registers when their value is no longer needed.

- Here is pseudocode (assumes nextRegInt has been declared in the class and initialized):

```
NextOpenRegId() returns String
   Declare String regIdString
   Set regIdString to "r" + nextRegInt
   Set nextRetInt to nextRegInt + 1
   Return regIdString
```



Generates a register id such as r1, r2, etc…

- The id class stores the variable's name.
- The value for this variable should be loaded into a register.
- If the Id node is on the left side of an assignment, then it will do an unnecessary load. We could code the assignment node so that it does not call code() on an Id if the Id node is the left side of an assignment statement.

```
class Id extends Expr {
    Declare String name
    Id Constructor (String name) {
        Set this.name to name
    }
    code() returns String {
        Declare String regId
        Set regId to NextOpenRegId()
        Declare String line
        Set line to "load " + regId + ", " + name
        Add line to end of assemblyCode
        Return regId
    }
}
```

Build a string for the load instruction. The member variable name has the variable to store to.

assemblyCode is a variable that holds the generated code. Can make this a List<String>. Each item is one line of code.

Return the register that is holding the value

AST Node Id Code Generation

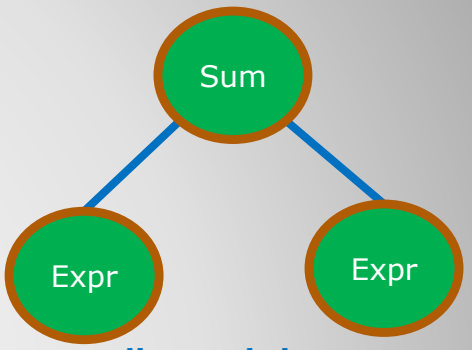
AST Node Sum Code Generation

- Recursively calls code for left and right sides.

```
class Sum extends Expr {
    Declare Expr lhs
    Declare Expr rhs
    Sum Constructor(Expr lhs, Expr rhs) {
        Set this.lhs to lhs
        Set this.rhs to rhs
    }

    code() returns String {
        Declare String line, lhsRegId, rhsRegId, resultRegId
        Set lhsRegId to lhs.code()
        Set rhsRegId to rhs.code()
        Set resultRegId to NextOpenRegId()
        Set line to "add " + lhsRegId + ", " + rhsRegId + ", " + resultRegId
        Add line to end of assemblyCode
        Return resultRegId
    }
}
```

The Sum class is used for operator (+) nodes in the AST



lhs and rhs are children of Sum

Get registers returned by child nodes

Build a string for the add instruction

- **End of Slides**

End of Slides