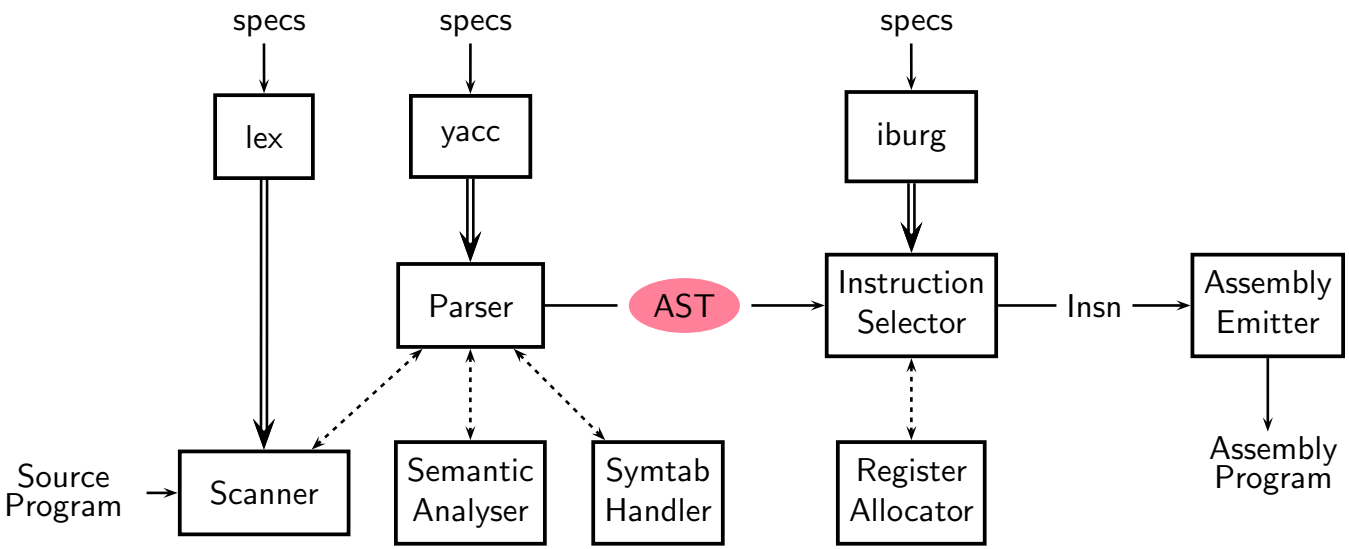
Part 7

Simple Compiler

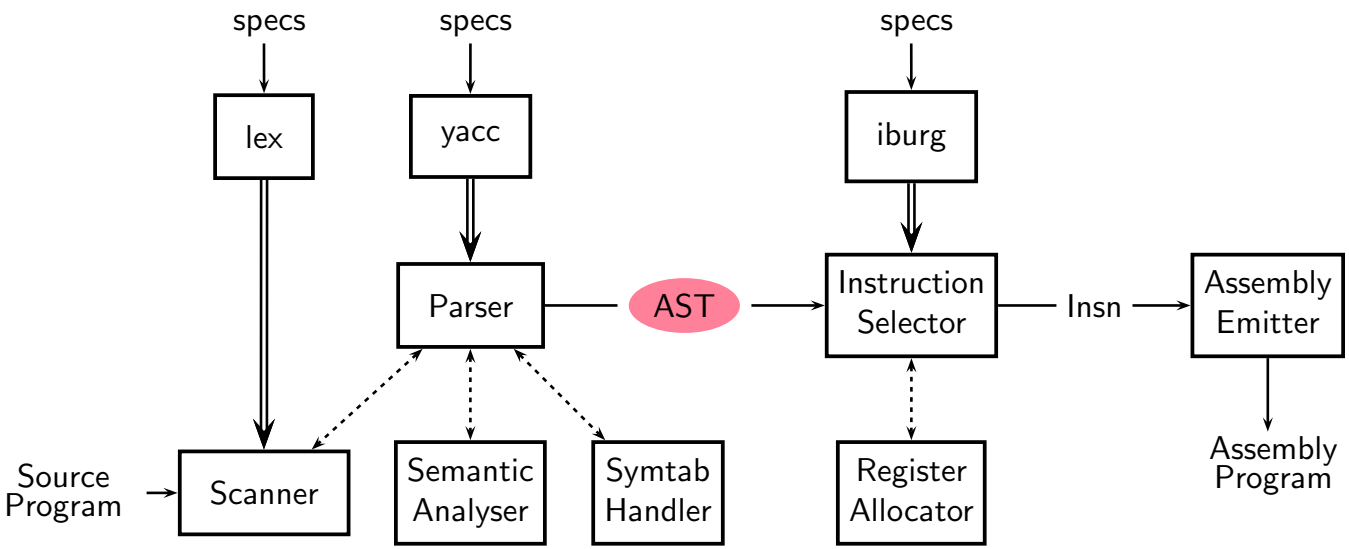
Compiler for Constant Expressions

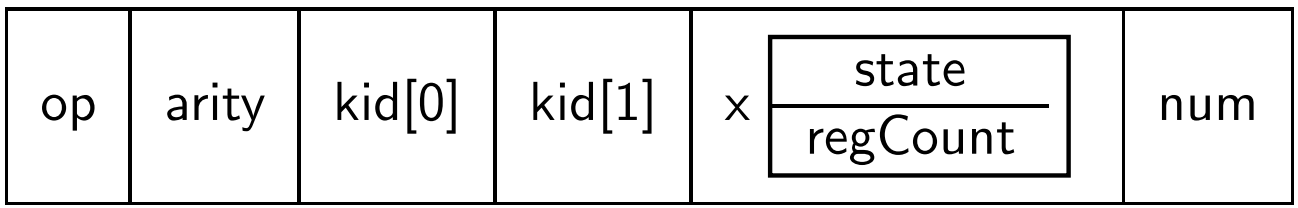
```
uday-laptop../ceCC -help

Usage: ceCC [options] [file]
Options:
  -help Show this help
  -tok   Show tokens in file.toks (or out.toks)
  -ast  Show abstract syntax tree in file.ast (or out.ast)
  -dst  Display abstract syntax tree using a postscript viewer
  -tc   Show a trace of the tree cover matching performed by the
        instruction selector in file.tc (or out.tc)
  -olr  Optimize local register usage through sethi-ullman
        numbering
  -src  Show local register count for each expression
  -insn Show instruction list in file.insn (or out.insn)
  -d    Demo version. Use stdout for the outputs instead of files
```


Type `ast`: Abstract Syntax Tree

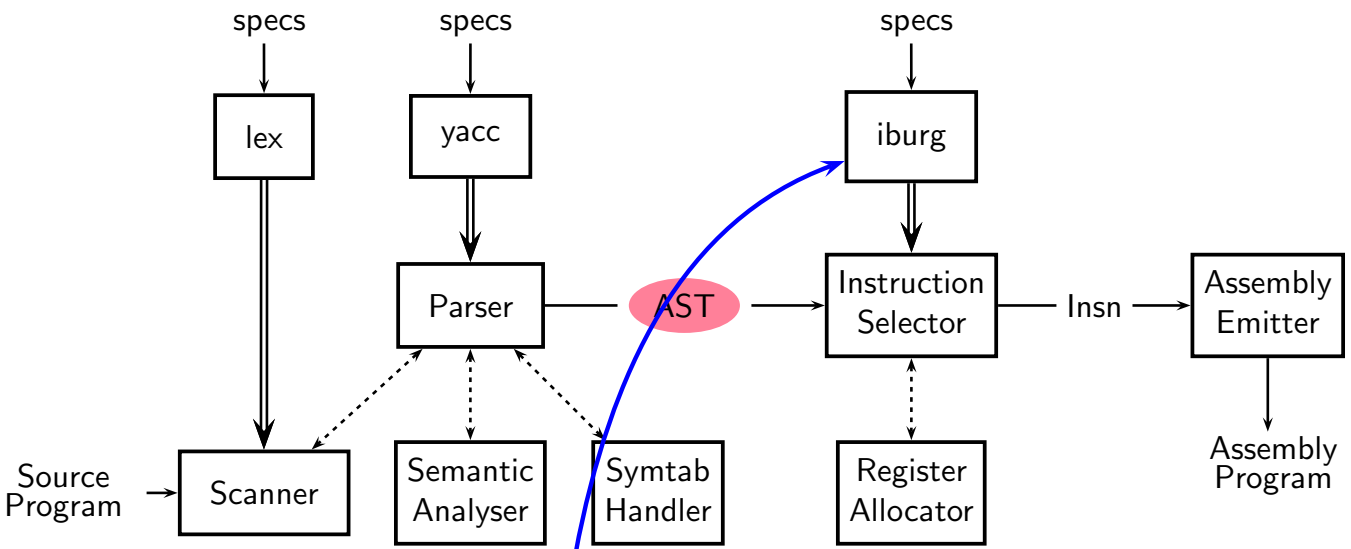


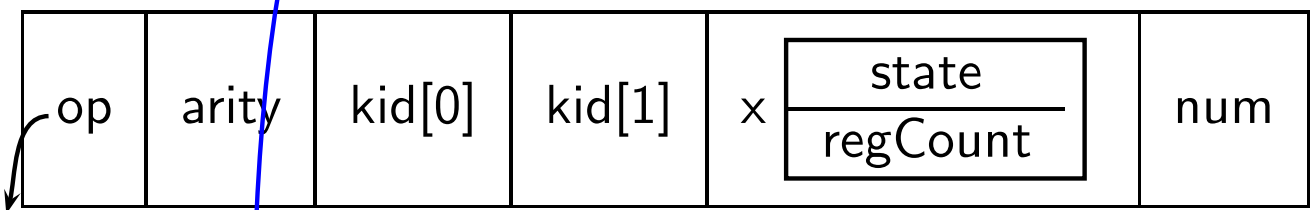
Type ast: Abstract Syntax Tree





Type `ast`: Abstract Syntax Tree

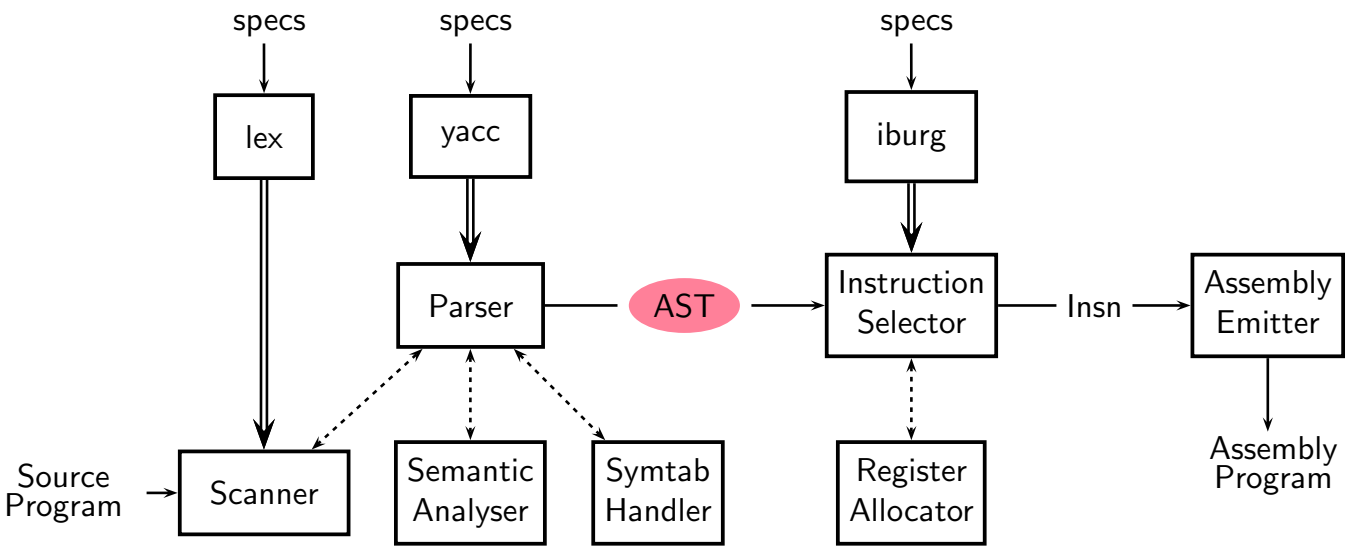


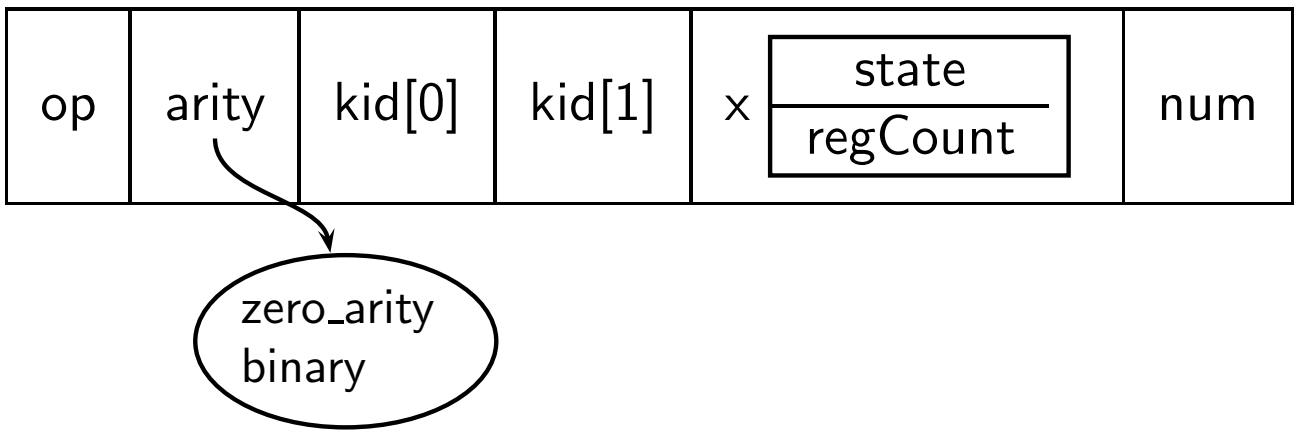


astPlusOp
astMultOp
astConstLeafOp

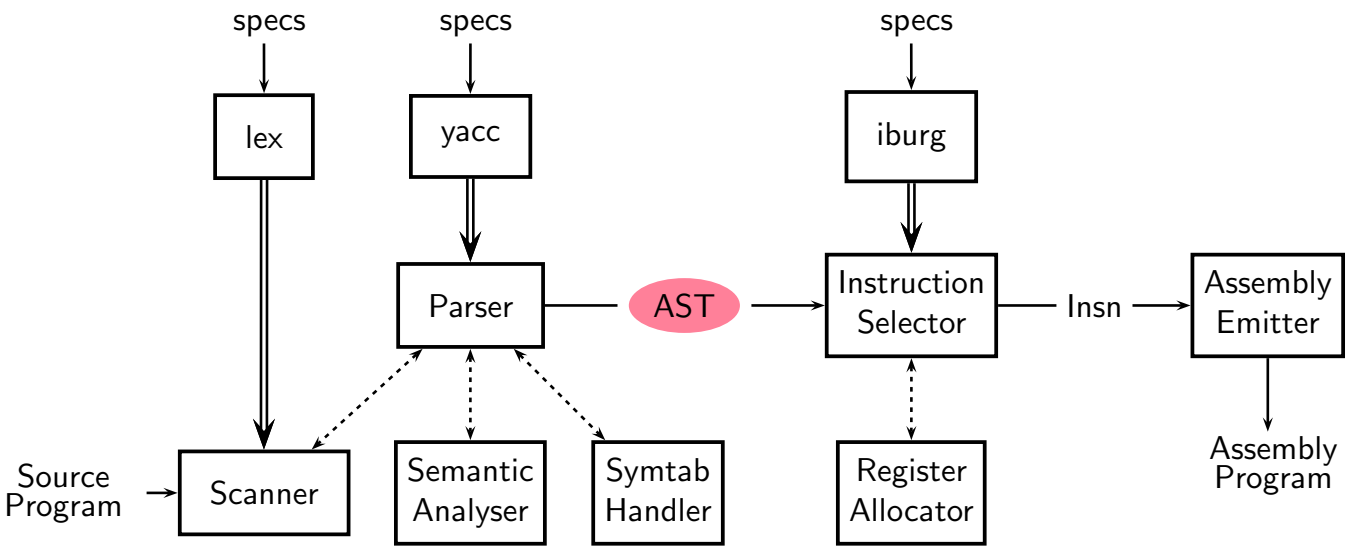
Expected as numbers by `iburg`.
Cannot be `#define` or `enum`

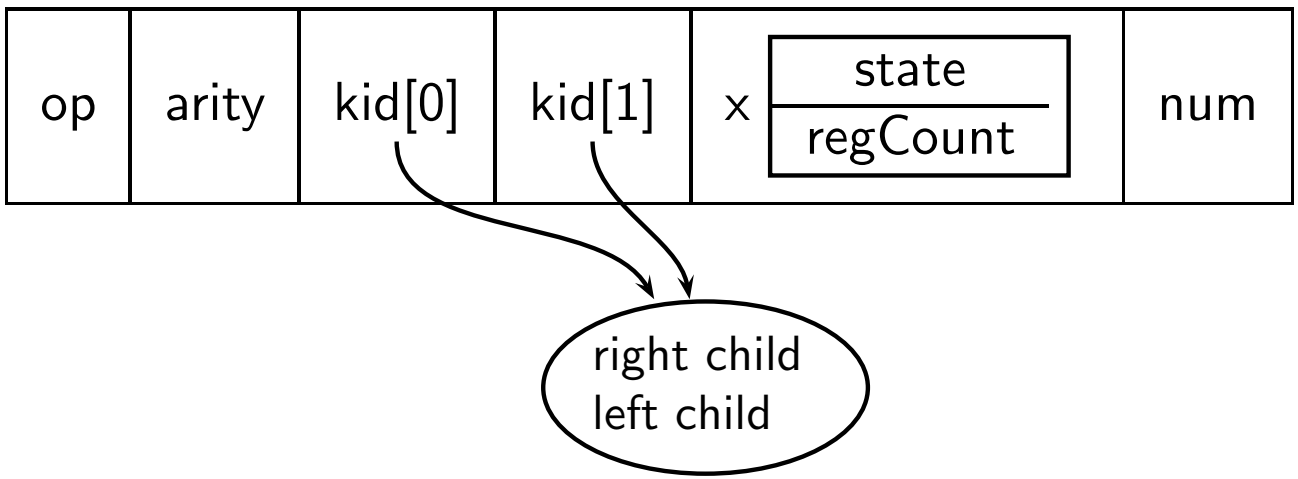
Type ast: Abstract Syntax Tree



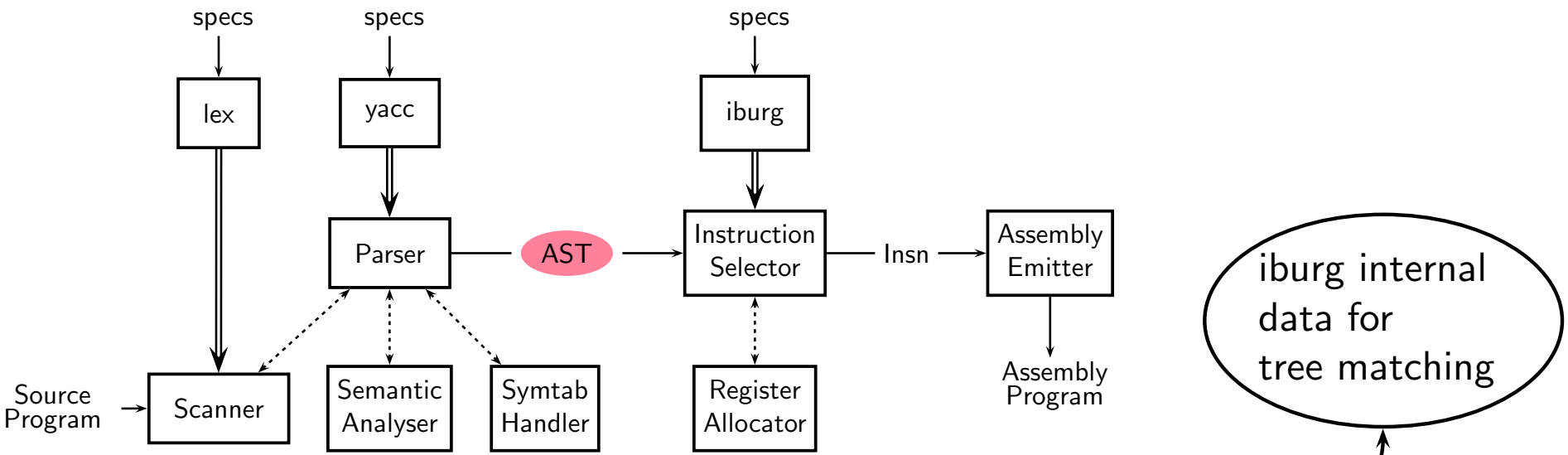


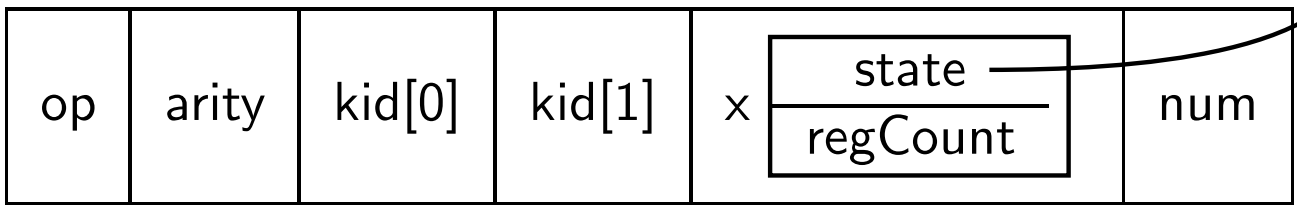
Type ast: Abstract Syntax Tree



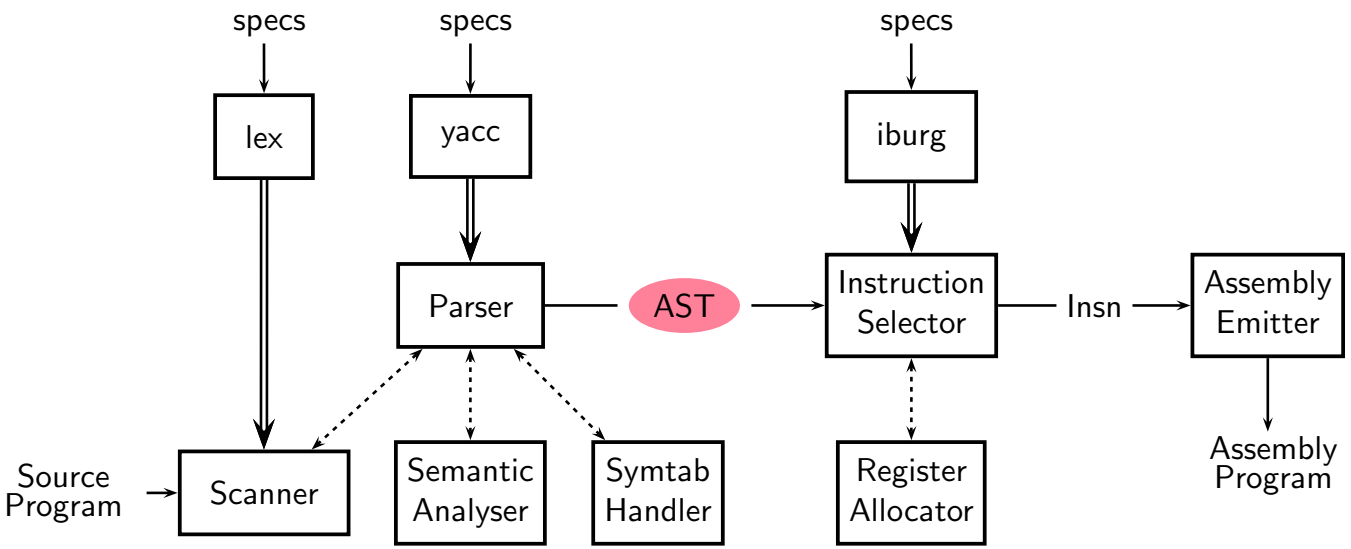


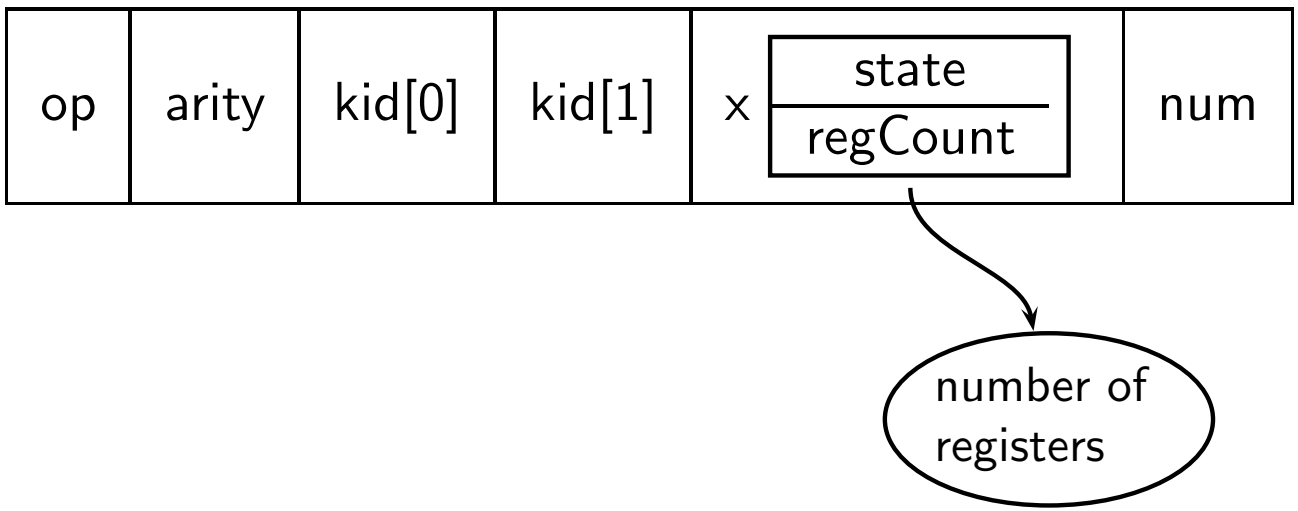
Type ast: Abstract Syntax Tree



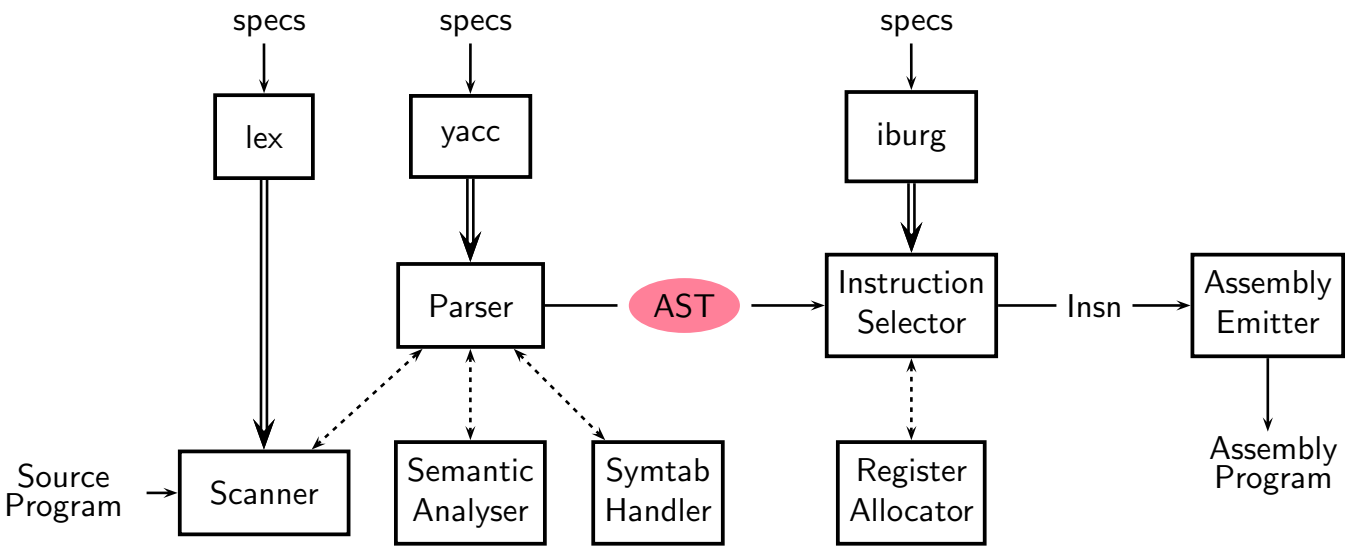


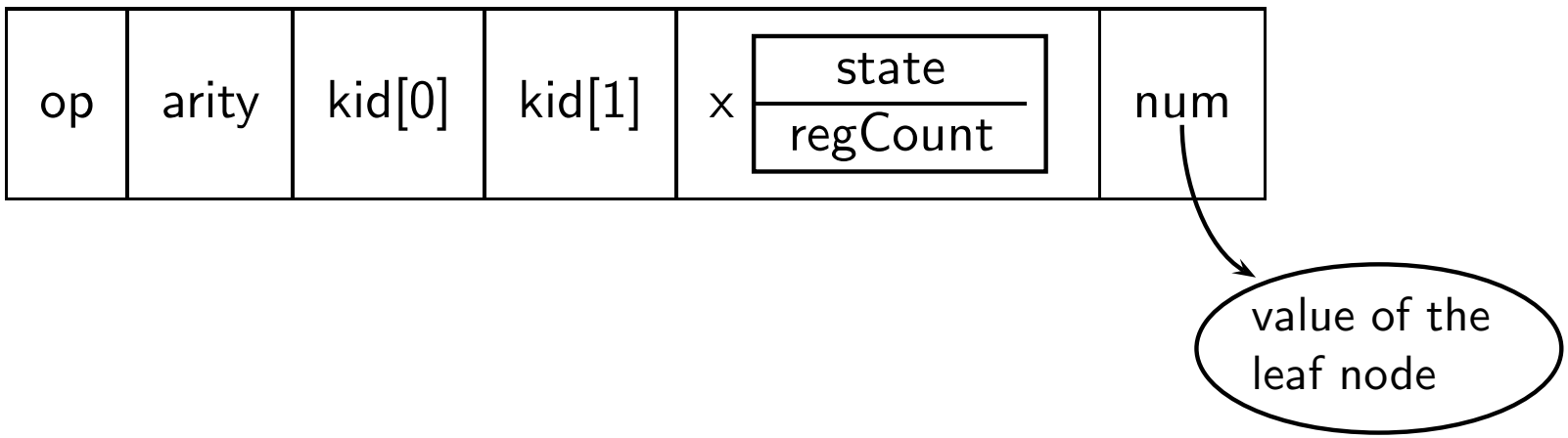
Type ast: Abstract Syntax Tree



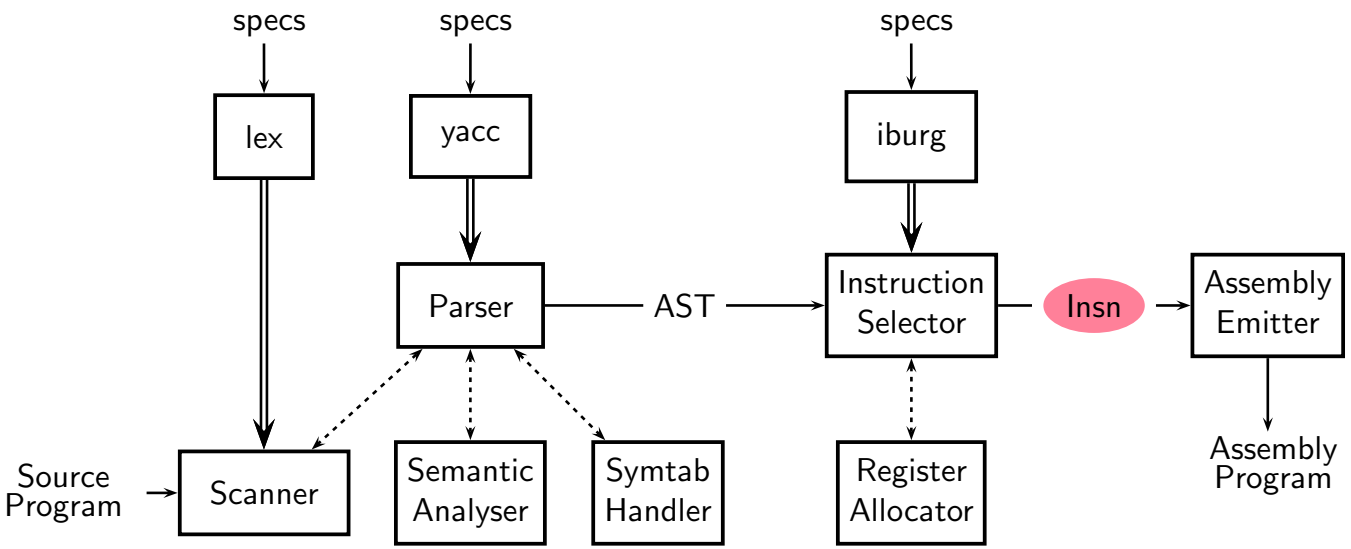


Type ast: Abstract Syntax Tree

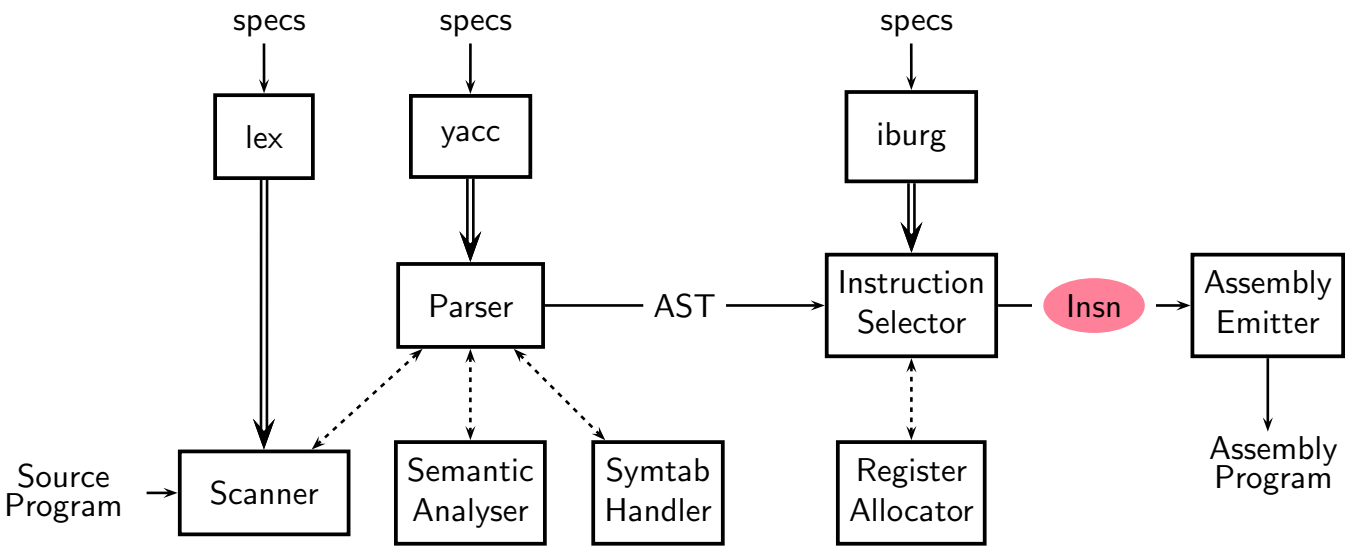




Quadruples for Instructions

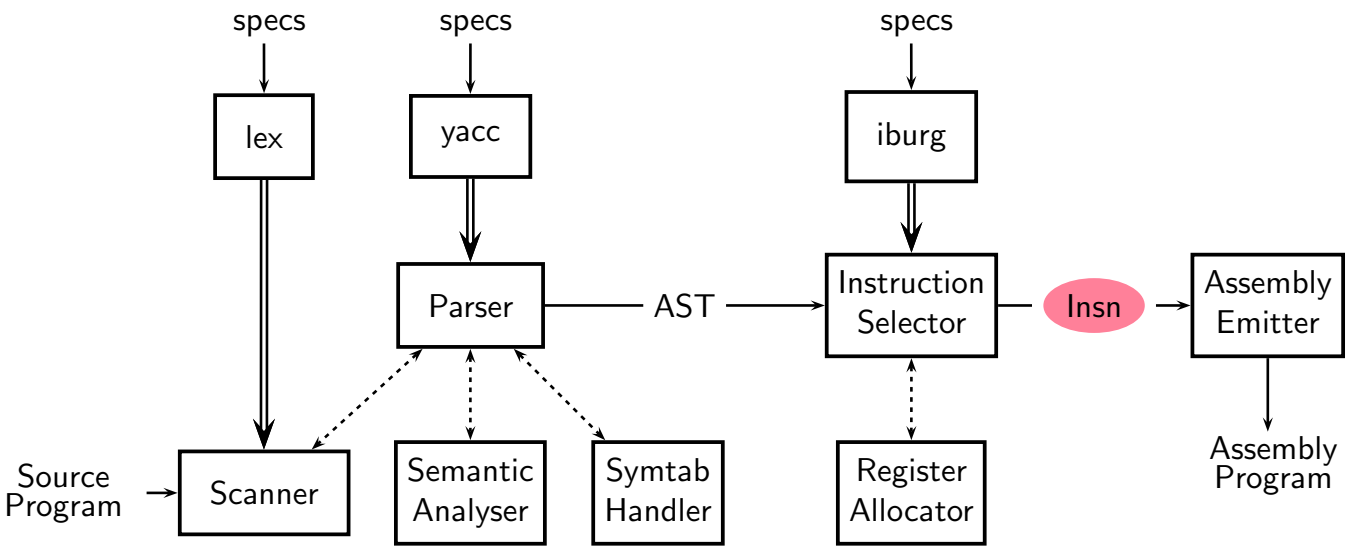


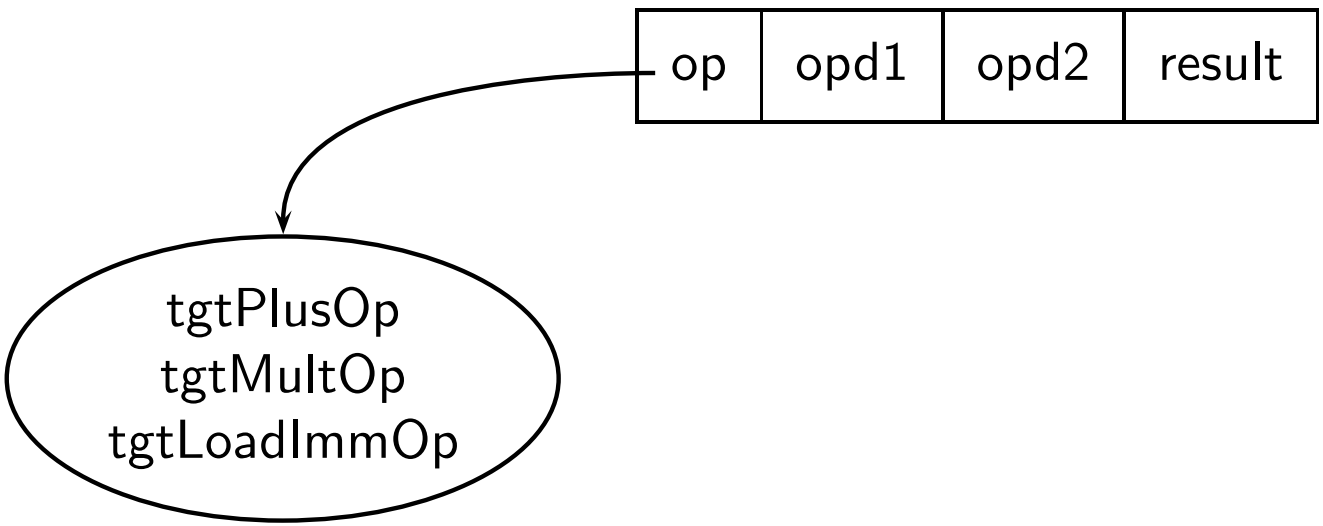
Quadruples for Instructions



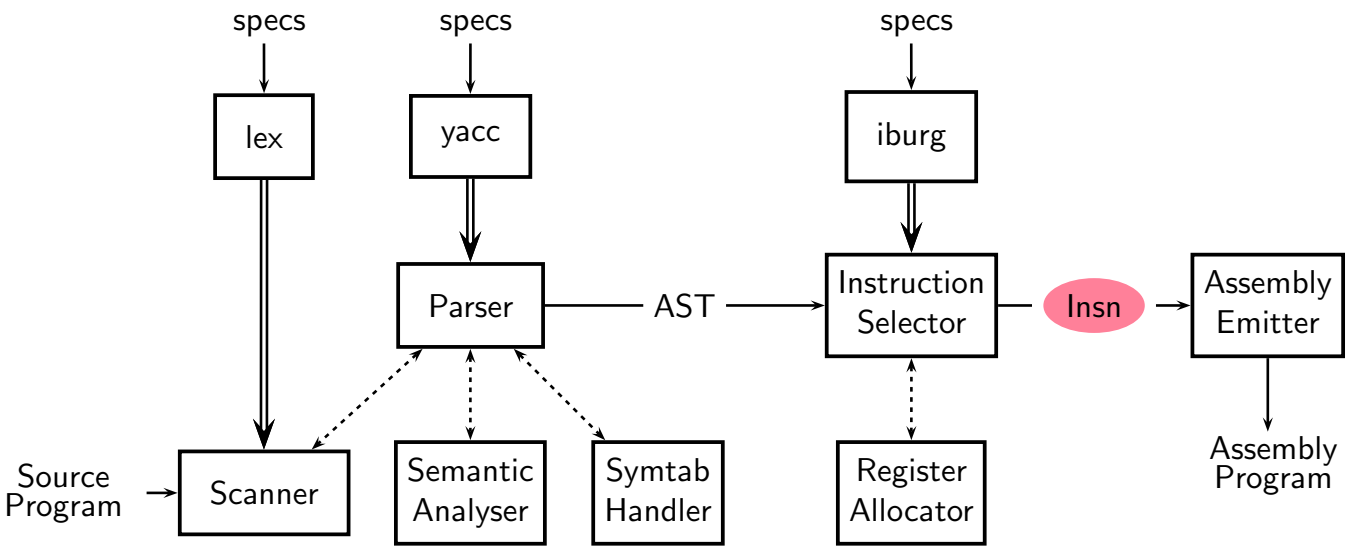
op	opd1	opd2	result

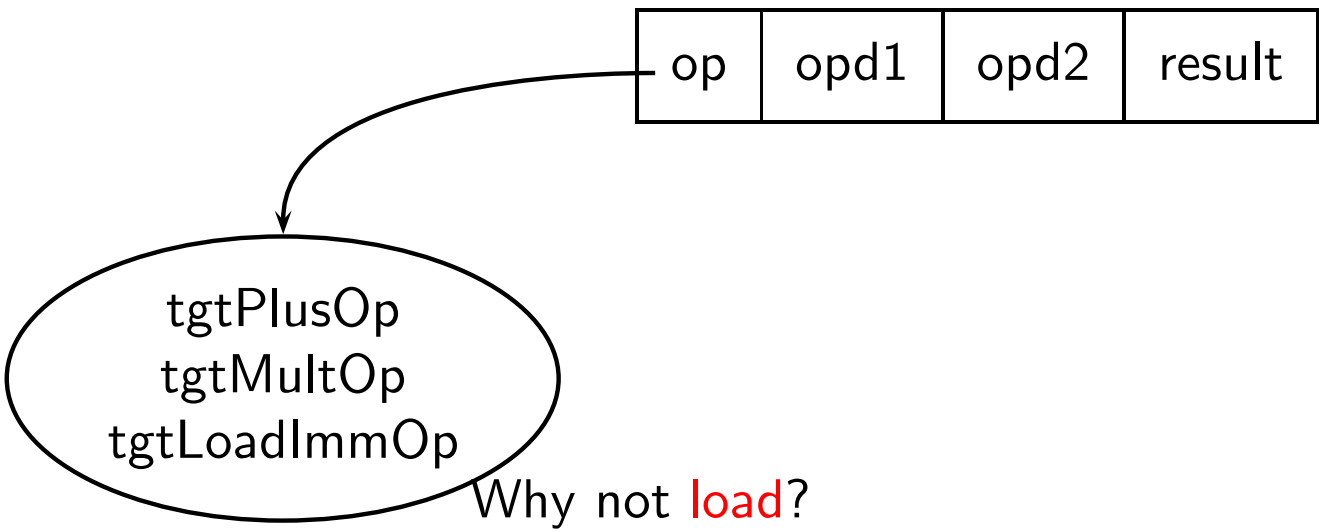
Quadruples for Instructions





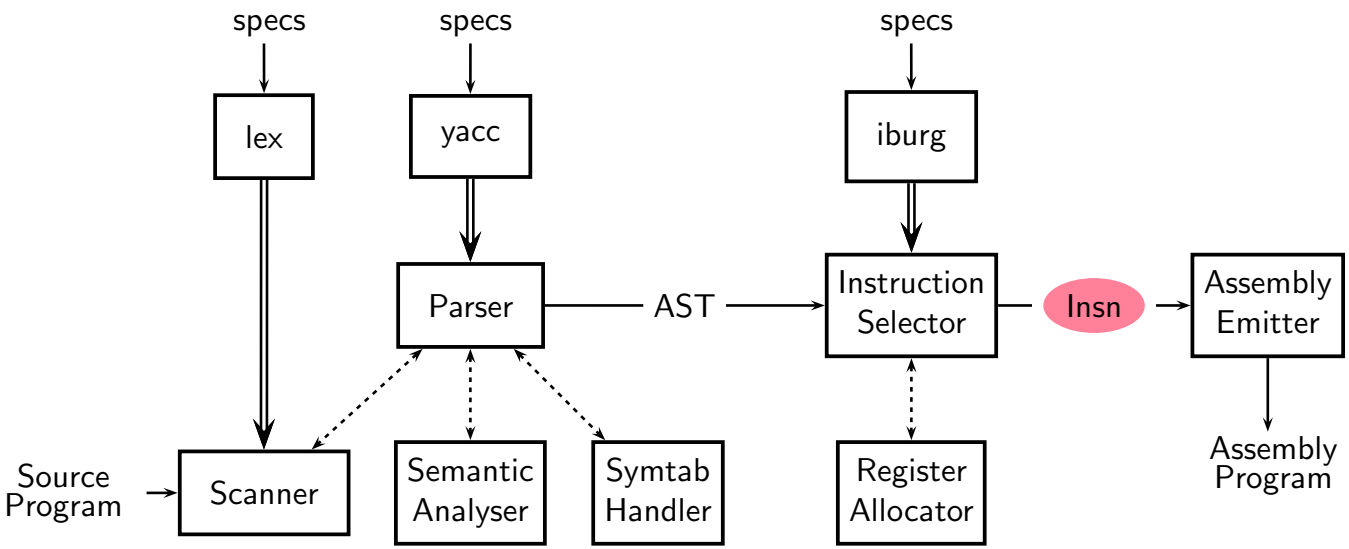
Quadruples for Instructions

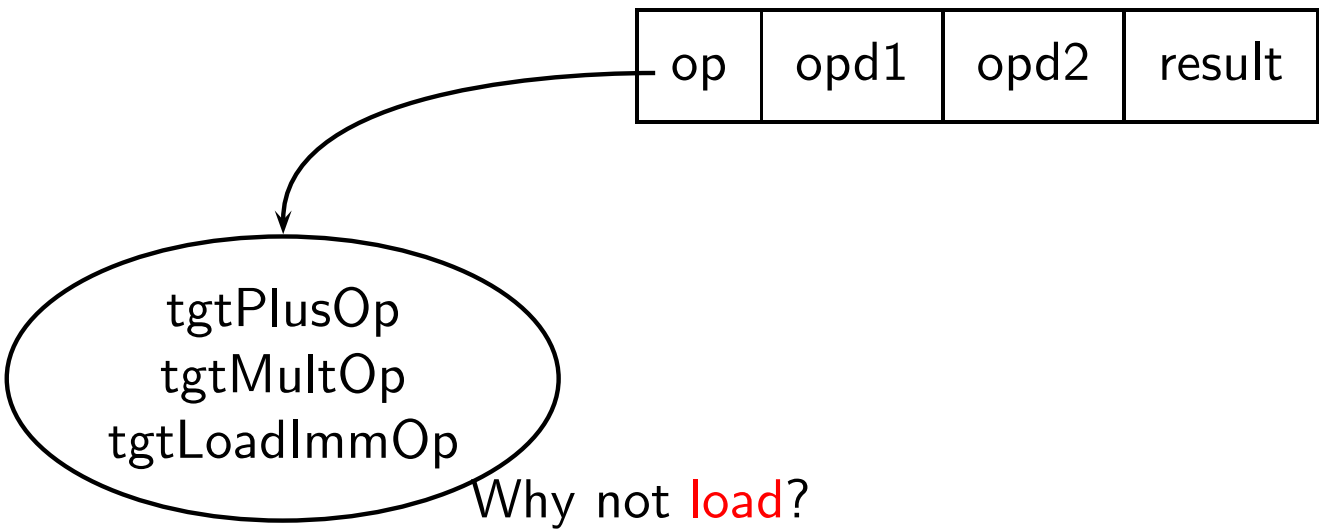




Why not load?

Quadruples for Instructions

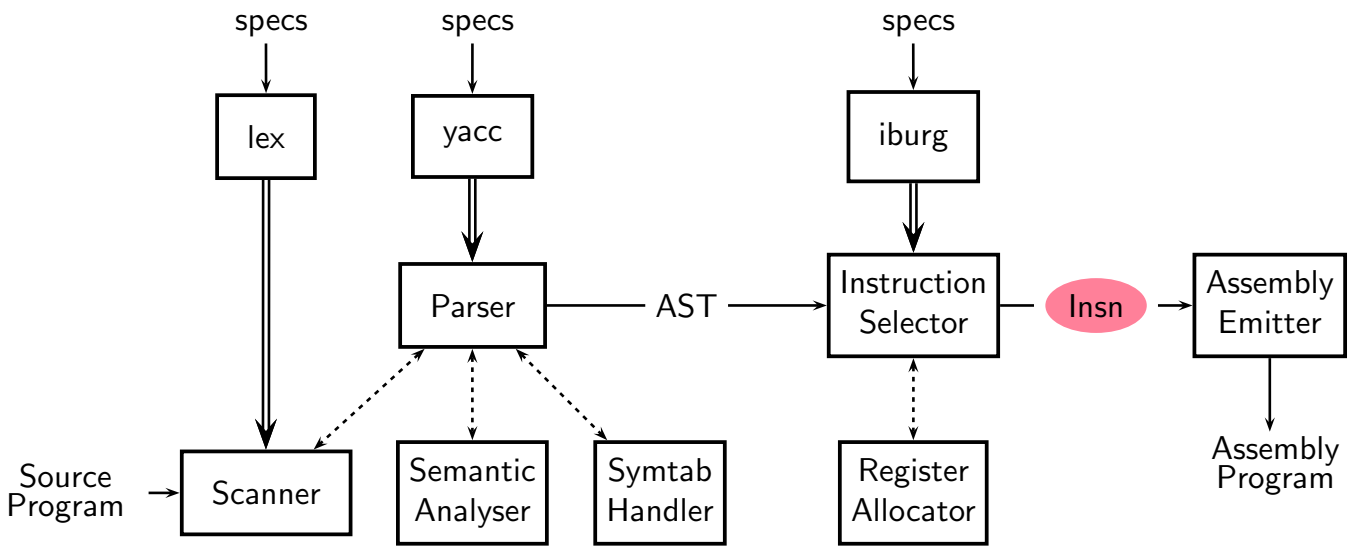


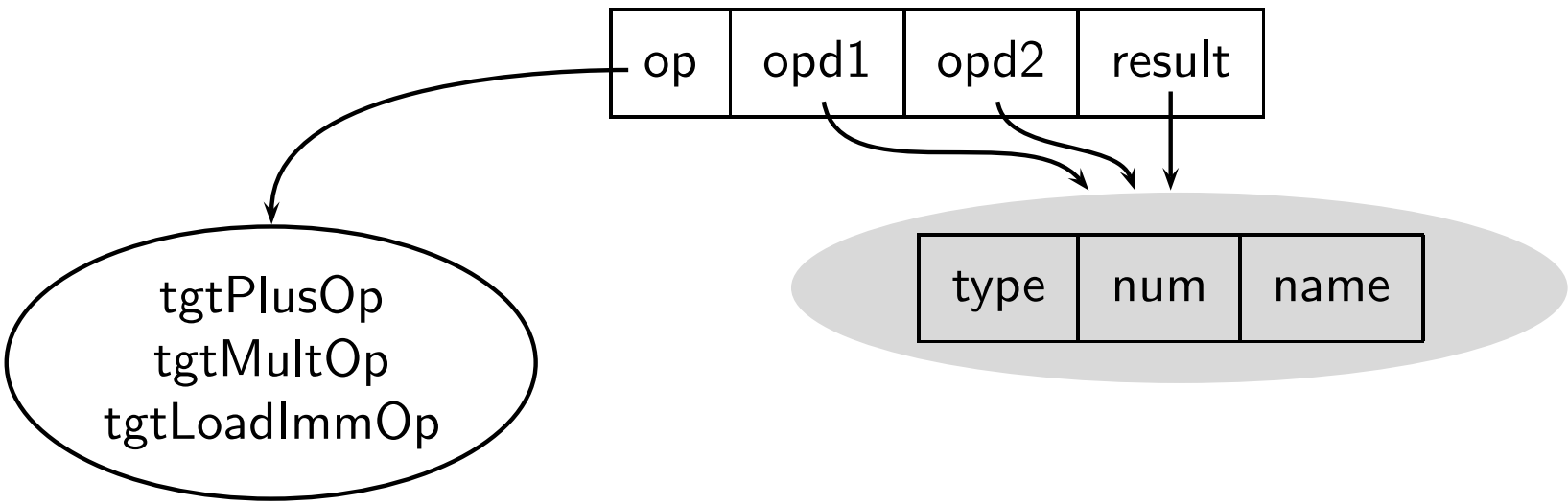


Why not load?

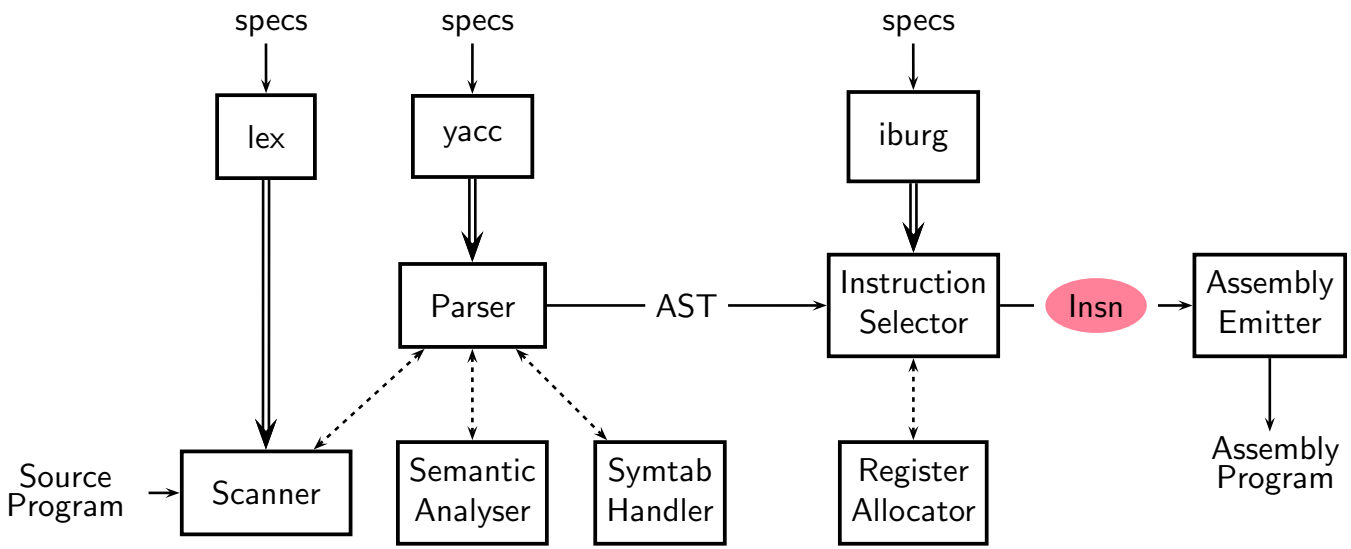
We don't have variables (i.e. locations)

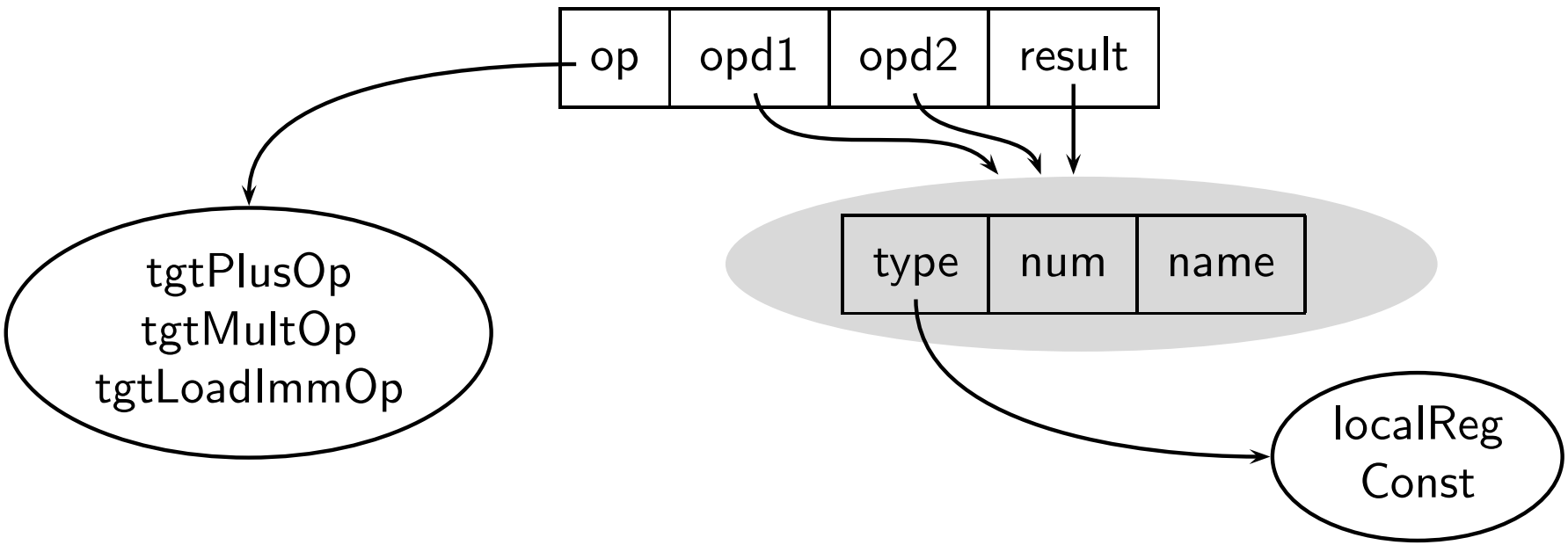
Quadruples for Instructions





Quadruples for Instructions





Using Tree Parsing for Instruction Selection

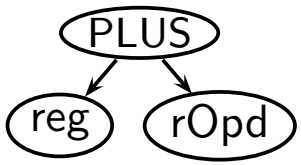
- Given instructions in the form of a tree grammar, iburg generates a program to *tile* a tree with minimum cost

Rule	Instruction	Semantics Replace tree By tree	Cost
reg:PLUS(reg,rOpd)	add		1
reg:MULT(reg,rOpd)	mult		1
reg:CONST	load immediate		1
rOpd:reg			0
rOpd:CONST			0

- Instruction selection is performed by the actions associated with a grammar rule

Using Tree Parsing for Instruction Selection

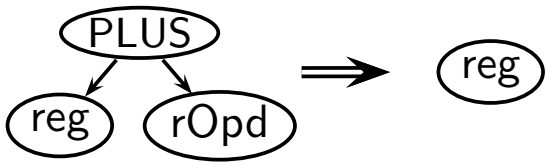
- Given instructions in the form of a tree grammar, iburg generates a program to *tile* a tree with minimum cost

Rule	Instruction	Semantics Replace tree By tree	Cost
reg:PLUS(reg,rOpd)	add		1
reg:MULT(reg,rOpd)	mult		1
reg:CONST	load immediate		1
rOpd:reg			0
rOpd:CONST			0

- Instruction selection is performed by the actions associated with a grammar rule

Using Tree Parsing for Instruction Selection

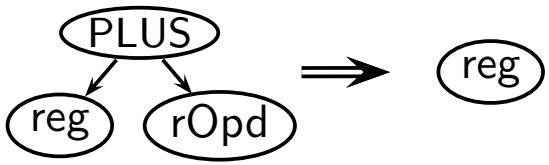
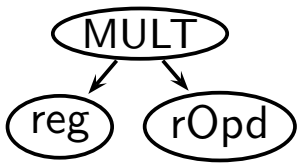
- Given instructions in the form of a tree grammar, iburg generates a program to *tile* a tree with minimum cost

Rule	Instruction	Semantics Replace tree By tree	Cost
reg:PLUS(reg,rOpd)	add		1
reg:MULT(reg,rOpd)	mult		1
reg:CONST	load immediate		1
rOpd:reg			0
rOpd:CONST			0

- Instruction selection is performed by the actions associated with a grammar rule

Using Tree Parsing for Instruction Selection

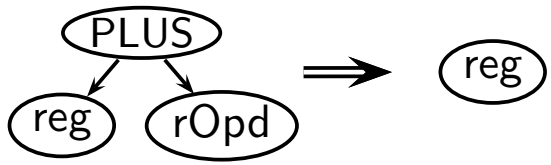
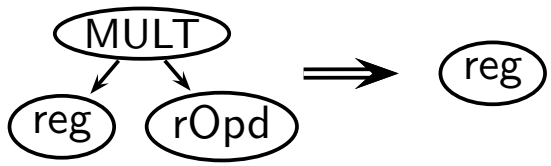
- Given instructions in the form of a tree grammar, iburg generates a program to *tile* a tree with minimum cost

Rule	Instruction	Semantics Replace tree By tree	Cost
reg:PLUS(reg,rOpd)	add		1
reg:MULT(reg,rOpd)	mult		1
reg:CONST	load immediate		1
rOpd:reg			0
rOpd:CONST			0

- Instruction selection is performed by the actions associated with a grammar rule

Using Tree Parsing for Instruction Selection

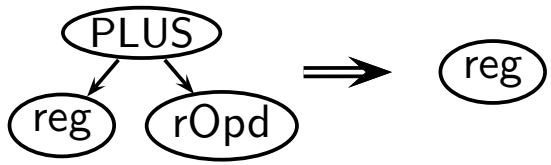
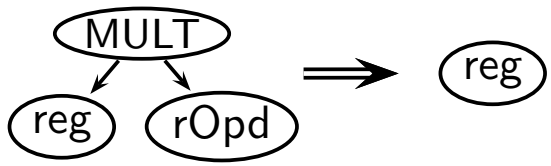
- Given instructions in the form of a tree grammar, iburg generates a program to *tile* a tree with minimum cost

Rule	Instruction	Semantics Replace tree By tree	Cost
reg:PLUS(reg,rOpd)	add		1
reg:MULT(reg,rOpd)	mult		1
reg:CONST	load immediate		1
rOpd:reg			0
rOpd:CONST			0

- Instruction selection is performed by the actions associated with a grammar rule

Using Tree Parsing for Instruction Selection

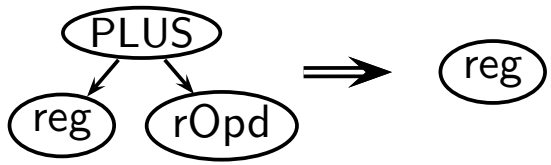
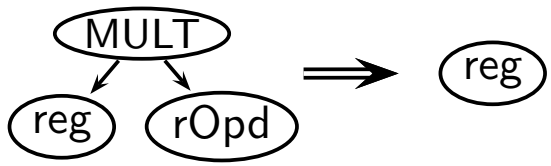
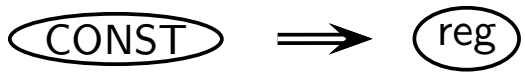
- Given instructions in the form of a tree grammar, iburg generates a program to *tile* a tree with minimum cost

Rule	Instruction	Semantics Replace tree By tree	Cost
reg:PLUS(reg,rOpd)	add		1
reg:MULT(reg,rOpd)	mult		1
reg:CONST	load immediate		1
rOpd:reg			0
rOpd:CONST			0

- Instruction selection is performed by the actions associated with a grammar rule

Using Tree Parsing for Instruction Selection

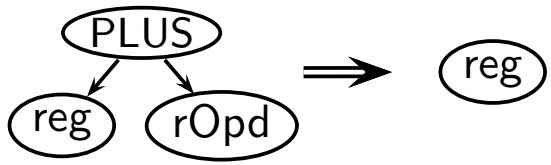
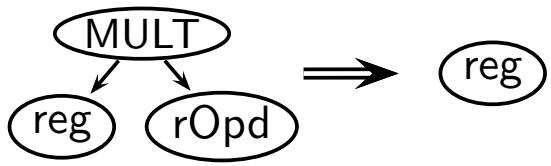
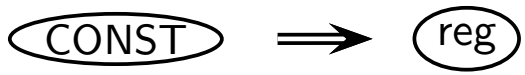
- Given instructions in the form of a tree grammar, iburg generates a program to *tile* a tree with minimum cost

Rule	Instruction	Semantics Replace tree By tree	Cost
reg:PLUS(reg,rOpd)	add		1
reg:MULT(reg,rOpd)	mult		1
reg:CONST	load immediate		1
rOpd:reg			0
rOpd:CONST			0

- Instruction selection is performed by the actions associated with a grammar rule

Using Tree Parsing for Instruction Selection

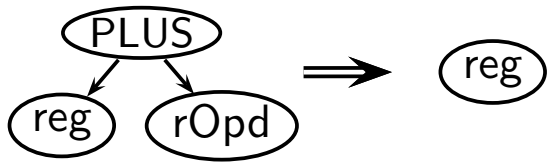
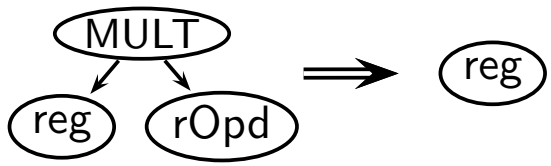
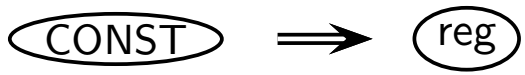
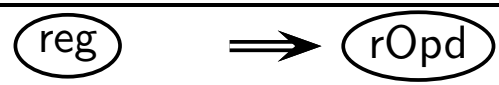
- Given instructions in the form of a tree grammar, iburg generates a program to *tile* a tree with minimum cost

Rule	Instruction	Semantics Replace tree By tree	Cost
reg:PLUS(reg,rOpd)	add		1
reg:MULT(reg,rOpd)	mult		1
reg:CONST	load immediate		1
rOpd:reg			0
rOpd:CONST			0

- Instruction selection is performed by the actions associated with a grammar rule

Using Tree Parsing for Instruction Selection

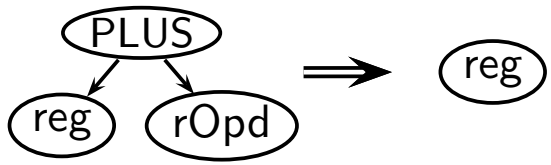
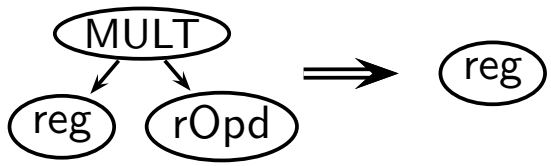
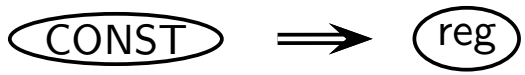
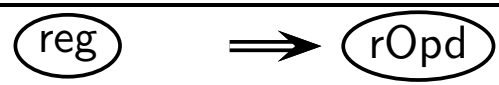
- Given instructions in the form of a tree grammar, iburg generates a program to *tile* a tree with minimum cost

Rule	Instruction	Semantics Replace tree By tree	Cost
reg:PLUS(reg,rOpd)	add		1
reg:MULT(reg,rOpd)	mult		1
reg:CONST	load immediate		1
rOpd:reg			0
rOpd:CONST			0

- Instruction selection is performed by the actions associated with a grammar rule

Using Tree Parsing for Instruction Selection

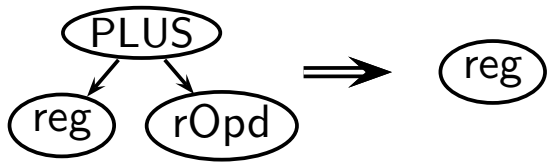
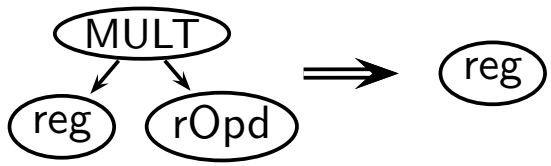
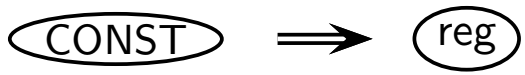
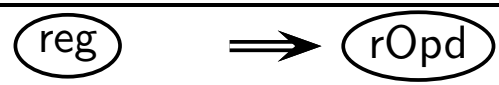
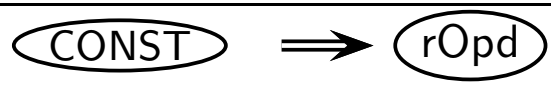
- Given instructions in the form of a tree grammar, iburg generates a program to *tile* a tree with minimum cost

Rule	Instruction	Semantics Replace tree By tree	Cost
reg:PLUS(reg,rOpd)	add		1
reg:MULT(reg,rOpd)	mult		1
reg:CONST	load immediate		1
rOpd:reg			0
rOpd:CONST			0

- Instruction selection is performed by the actions associated with a grammar rule

Using Tree Parsing for Instruction Selection

- Given instructions in the form of a tree grammar, iburg generates a program to *tile* a tree with minimum cost

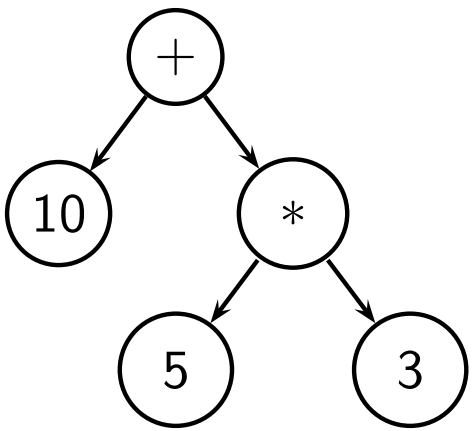
Rule	Instruction	Semantics Replace tree ⟹ By tree	Cost
reg:PLUS(reg,rOpd)	add		1
reg:MULT(reg,rOpd)	mult		1
reg:CONST	load immediate		1
rOpd:reg			0
rOpd:CONST			0

- Instruction selection is performed by the actions associated with a grammar rule

An Example of Tree Parsing

- Consider the Input `10 + 5 * 3`

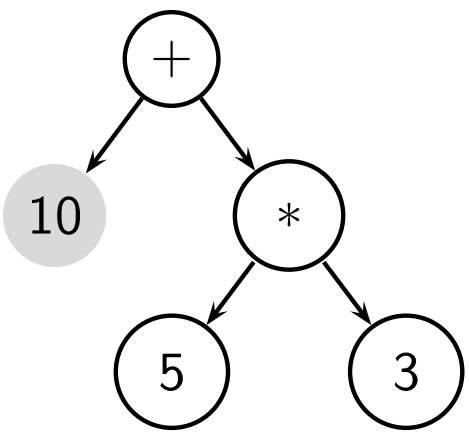
Subject Tree | Primitive Tree Matched | Generated Instructions

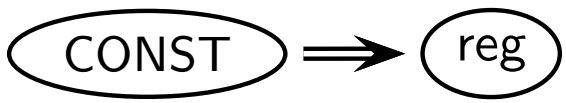


An Example of Tree Parsing

- Consider the Input `10 + 5 * 3`

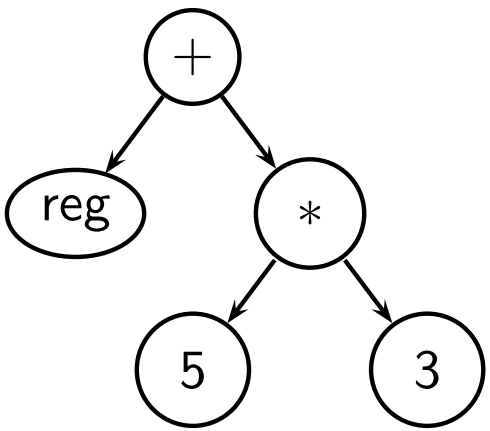
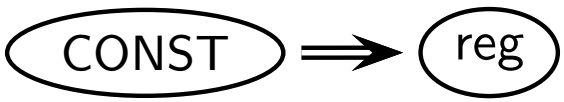
Subject Tree | Primitive Tree Matched | Generated Instructions





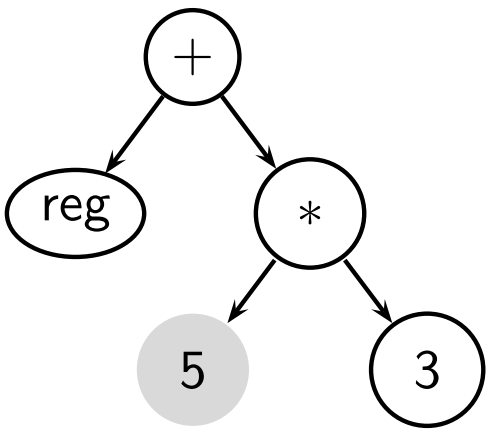
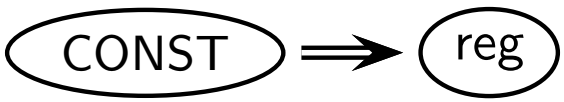
An Example of Tree Parsing

- Consider the Input `10 + 5 * 3`

Subject Tree	Primitive Tree Matched	Generated Instructions
		`li $t0, 10`

An Example of Tree Parsing

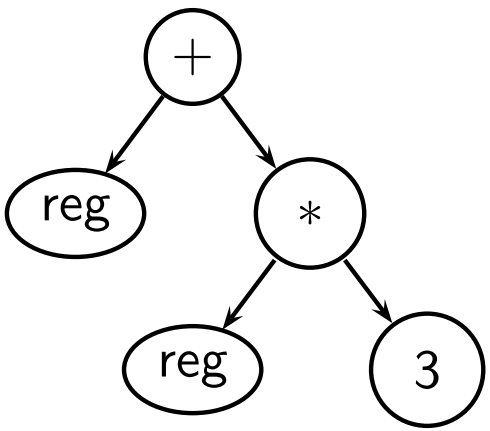
- Consider the Input `10 + 5 * 3`

Subject Tree	Primitive Tree Matched	Generated Instructions
		`li $t0, 10`

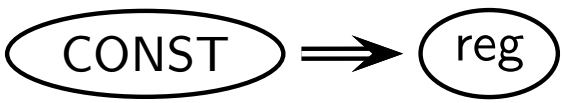
An Example of Tree Parsing

- Consider the Input `10 + 5 * 3`

Subject Tree



Primitive Tree Matched



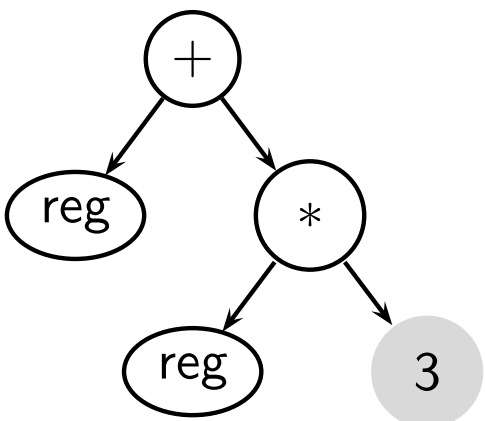
Generated Instructions

```
li $t0, 10
li $t1, 5
```

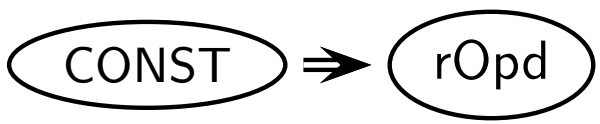
An Example of Tree Parsing

- Consider the Input `10 + 5 * 3`

Subject Tree



Primitive Tree Matched



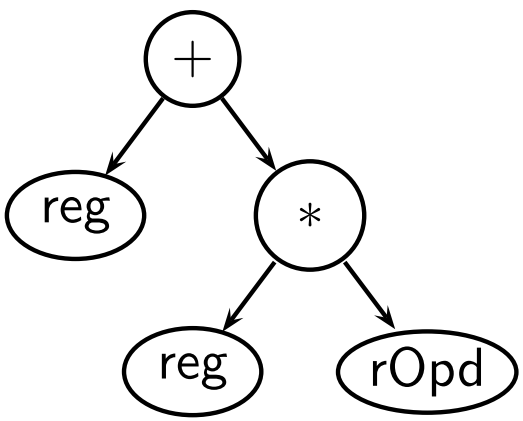
Generated Instructions

```
li $t0, 10
li $t1, 5
```

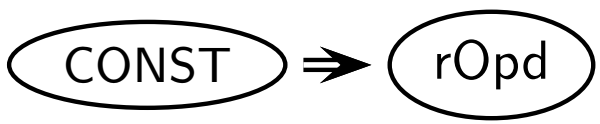
An Example of Tree Parsing

- Consider the Input `10 + 5 * 3`

Subject Tree



Primitive Tree Matched



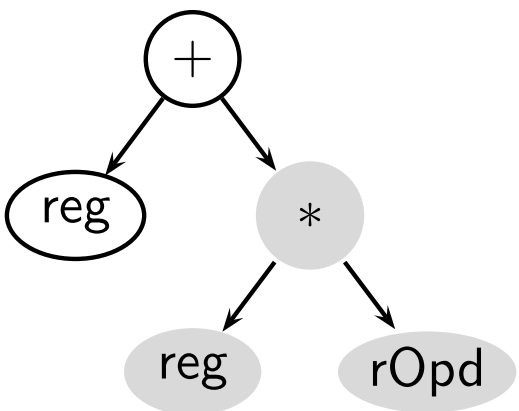
Generated Instructions

```
li $t0, 10
li $t1, 5
```

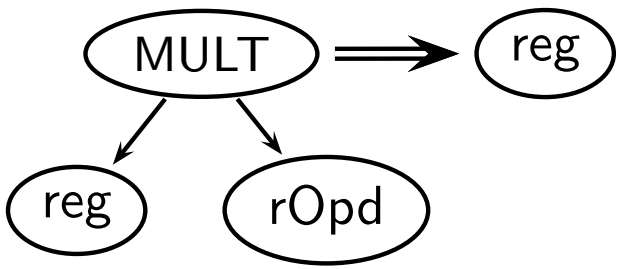

An Example of Tree Parsing

- Consider the Input `10 + 5 * 3`

Subject Tree



Primitive Tree Matched



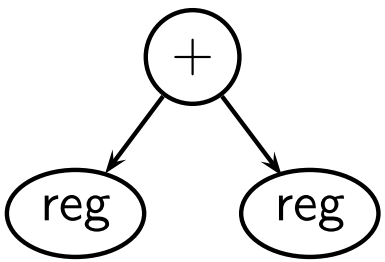
Generated Instructions

```
li $t0, 10
li $t1, 5
```

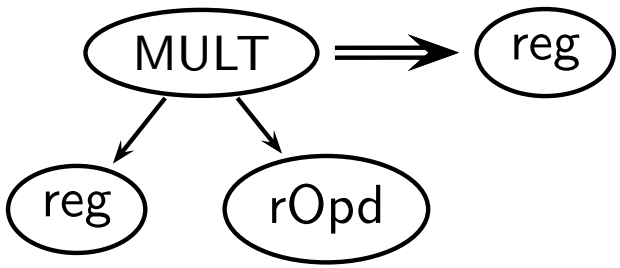
An Example of Tree Parsing

- Consider the Input `10 + 5 * 3`

Subject Tree



Primitive Tree Matched



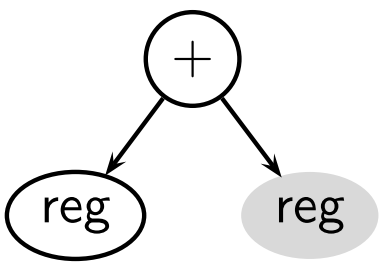
Generated Instructions

```
li $t0, 10
li $t1, 5
mul $t2, $t1, 3
```

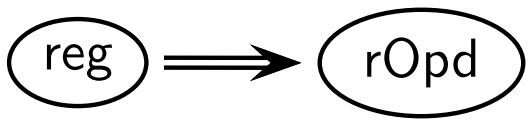

An Example of Tree Parsing

- Consider the Input `10 + 5 * 3`

Subject Tree



Primitive Tree Matched



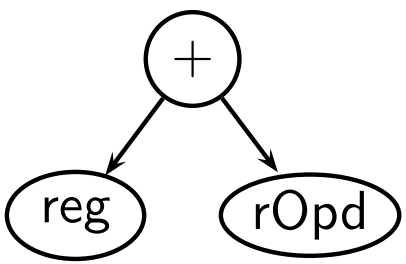
Generated Instructions

```
li $t0, 10
li $t1, 5
mul $t2, $t1, 3
```

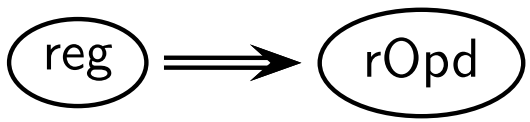

An Example of Tree Parsing

- Consider the Input `10 + 5 * 3`

Subject Tree



Primitive Tree Matched



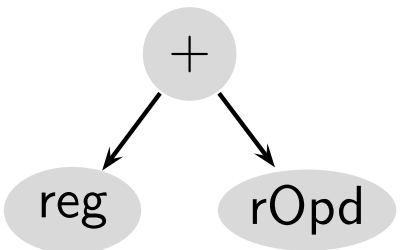
Generated Instructions

```
li $t0, 10
li $t1, 5
mul $t2, $t1, 3
```

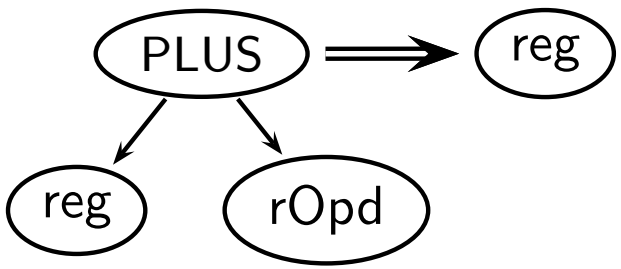
An Example of Tree Parsing

- Consider the Input `10 + 5 * 3`

Subject Tree



Primitive Tree Matched



Generated Instructions

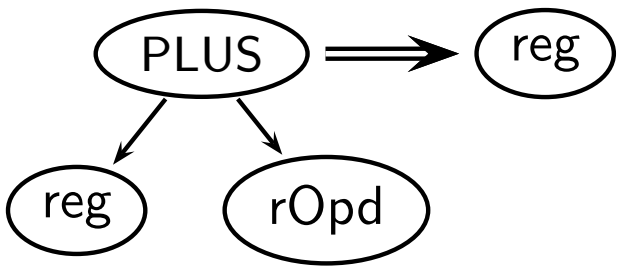
```
li $t0, 10
li $t1, 5
mul $t2, $t1, 3
```

An Example of Tree Parsing

- Consider the Input `10 + 5 * 3`

Subject Tree

Primitive Tree Matched



Generated Instructions

```
li $t0, 10
li $t1, 5
mul $t2, $t1, 3
add $t3, $t0, $t2
```


Use of Ambiguous Grammars

- Grammars used to represent instructions are usually ambiguous.

Use of Ambiguous Grammars

- Grammars used to represent instructions are usually ambiguous.
- Consider the following two parses

Use of Ambiguous Grammars

- Grammars used to represent instructions are usually ambiguous.
- Consider the following two parses

Input	Reduction	Instruction emitted
PLUS(c_1, c_2)	$\Rightarrow$ PLUS(reg, c_2)	

Use of Ambiguous Grammars

- Grammars used to represent instructions are usually ambiguous.
- Consider the following two parses

Input	Reduction	Instruction emitted
PLUS(c_1, c_2)	$\Rightarrow$ PLUS(reg, c_2)	`li $t0,` c_1

Use of Ambiguous Grammars

- Grammars used to represent instructions are usually ambiguous.
- Consider the following two parses

Input	Reduction	Instruction emitted
PLUS(c_1, c_2)	$\Rightarrow$ PLUS(reg, c_2)	`li $t0,` c_1
	$\Rightarrow$ PLUS(reg, rOpd)	

Use of Ambiguous Grammars

- Grammars used to represent instructions are usually ambiguous.
- Consider the following two parses

Input	Reduction	Instruction emitted
PLUS(c_1, c_2)	$\Rightarrow$ PLUS(reg, c_2)	`li $t0,` c_1
	$\Rightarrow$ PLUS(reg, rOpd)	
	$\Rightarrow$ reg	

Use of Ambiguous Grammars

- Grammars used to represent instructions are usually ambiguous.
- Consider the following two parses

Input	Reduction	Instruction emitted
PLUS(c_1, c_2)	$\Rightarrow$ PLUS(reg, c_2)	`li $t0,` c_1
	$\Rightarrow$ PLUS(reg, rOpd)	
	$\Rightarrow$ reg	`add $t1, $t0,` c_2

Use of Ambiguous Grammars

- Grammars used to represent instructions are usually ambiguous.
- Consider the following two parses

Input	Reduction	Instruction emitted
PLUS(c_1, c_2)	$\Rightarrow$ PLUS(reg, c_2)	`li $t0,` c_1
	$\Rightarrow$ PLUS(reg, rOpd)	
	$\Rightarrow$ reg	`add $t1, $t0,` c_2

Input	Reduction	Instruction emitted
PLUS(c_1, c_2)	$\Rightarrow$ PLUS(reg, c_2)	

Use of Ambiguous Grammars

- Grammars used to represent instructions are usually ambiguous.
- Consider the following two parses

Input	Reduction	Instruction emitted
PLUS(c_1, c_2)	$\Rightarrow$ PLUS(reg, c_2)	`li $t0,` c_1
	$\Rightarrow$ PLUS(reg, rOpd)	
	$\Rightarrow$ reg	`add $t1, $t0,` c_2

Input	Reduction	Instruction emitted
PLUS(c_1, c_2)	$\Rightarrow$ PLUS(reg, c_2)	`li $t0,` c_1

Use of Ambiguous Grammars

- Grammars used to represent instructions are usually ambiguous.
- Consider the following two parses

Input	Reduction	Instruction emitted
PLUS(c_1, c_2)	$\Rightarrow$ PLUS(reg, c_2)	`li $t0,` c_1
	$\Rightarrow$ PLUS(reg, rOpd)	
	$\Rightarrow$ reg	`add $t1, $t0,` c_2

Input	Reduction	Instruction emitted
PLUS(c_1, c_2)	$\Rightarrow$ PLUS(reg, c_2)	`li $t0,` c_1
	$\Rightarrow$ PLUS(reg, reg)	

Use of Ambiguous Grammars

- Grammars used to represent instructions are usually ambiguous.
- Consider the following two parses

Input	Reduction	Instruction emitted
PLUS(c_1, c_2)	$\Rightarrow$ PLUS(reg, c_2)	`li $t0,` c_1
	$\Rightarrow$ PLUS(reg, rOpd)	
	$\Rightarrow$ reg	`add $t1, $t0,` c_2

Input	Reduction	Instruction emitted
PLUS(c_1, c_2)	$\Rightarrow$ PLUS(reg, c_2)	`li $t0,` c_1
	$\Rightarrow$ PLUS(reg, reg)	`li $t1,` c_2

Use of Ambiguous Grammars

- Grammars used to represent instructions are usually ambiguous.
- Consider the following two parses

Input	Reduction	Instruction emitted
PLUS(c_1, c_2)	$\Rightarrow$ PLUS(reg, c_2)	`li $t0,` c_1
	$\Rightarrow$ PLUS(reg, rOpd)	
	$\Rightarrow$ reg	`add $t1, $t0,` c_2

Input	Reduction	Instruction emitted
PLUS(c_1, c_2)	$\Rightarrow$ PLUS(reg, c_2)	`li $t0,` c_1
	$\Rightarrow$ PLUS(reg, reg)	`li $t1,` c_2
	$\Rightarrow$ PLUS(reg, rOpd)	

Use of Ambiguous Grammars

- Grammars used to represent instructions are usually ambiguous.
- Consider the following two parses

Input	Reduction	Instruction emitted
PLUS(c_1, c_2)	$\Rightarrow$ PLUS(reg, c_2)	`li $t0,` c_1
	$\Rightarrow$ PLUS(reg, rOpd)	
	$\Rightarrow$ reg	`add $t1, $t0,` c_2

Input	Reduction	Instruction emitted
PLUS(c_1, c_2)	$\Rightarrow$ PLUS(reg, c_2)	`li $t0,` c_1
	$\Rightarrow$ PLUS(reg, reg)	`li $t1,` c_2
	$\Rightarrow$ PLUS(reg, rOpd)	
	$\Rightarrow$ reg	

Use of Ambiguous Grammars

- Grammars used to represent instructions are usually ambiguous.
- Consider the following two parses

Input	Reduction	Instruction emitted
PLUS(c_1, c_2)	$\Rightarrow$ PLUS(reg, c_2)	`li $t0,` c_1
	$\Rightarrow$ PLUS(reg, rOpd)	
	$\Rightarrow$ reg	`add $t1, $t0,` c_2

Input	Reduction	Instruction emitted
PLUS(c_1, c_2)	$\Rightarrow$ PLUS(reg, c_2)	`li $t0,` c_1
	$\Rightarrow$ PLUS(reg, reg)	`li $t1,` c_2
	$\Rightarrow$ PLUS(reg, rOpd)	
	$\Rightarrow$ reg	`add $t2, $t0, $t1`

Use of Ambiguous Grammars

- Grammars used to represent instructions are usually ambiguous.
- Consider the following two parses

Input	Reduction	Instruction emitted
PLUS(c_1, c_2)	⇒ PLUS(reg, c_2)	`li $t0,` c_1
	⇒ PLUS(reg, rOpd)	
	⇒ reg	`add $t1, $t0,` c_2

Input	Reduction	Instruction emitted
PLUS(c_1, c_2)	⇒ PLUS(reg, c_2)	`li $t0,` c_1
	⇒ PLUS(reg, reg)	`li $t1,` c_2
	⇒ PLUS(reg, rOpd)	
	⇒ reg	`add $t2, $t0, $t1`

- Cost associated with instructions is used to select "better" parse.

Actions for Instruction Selection

- Task: Having identified a sub-tree, generate instructions for computing that sub-tree
- Recursive function

  ```
  codeForSubjectTree *
  gen_Inst_List_for_Subject_Tree(ast * p, int goalnt)
  ```

- If p is a sub-tree OP(leftOpd, rightOpd):

Actions for Instruction Selection

- Task: Having identified a sub-tree, generate instructions for computing that sub-tree
- Recursive function

  ```
  codeForSubjectTree *
  gen_Inst_List_for_Subject_Tree(ast * p, int goalnt)
  ```

- If p is a sub-tree OP(leftOpd, rightOpd):
  - Generate instructions for the subtree corresponding to the goal non-terminal leftOpd

Actions for Instruction Selection

- Task: Having identified a sub-tree, generate instructions for computing that sub-tree
- Recursive function

  ```
  codeForSubjectTree *
  gen_Inst_List_for_Subject_Tree(ast * p, int goalnt)
  ```

- If p is a sub-tree OP(leftOpd, rightOpd):
  - Generate instructions for the subtree corresponding to the goal non-terminal leftOpd
  - Generate instructions for the subtree corresponding to the goal non-terminal rightOpd

Actions for Instruction Selection

- Task: Having identified a sub-tree, generate instructions for computing that sub-tree
- Recursive function

  ```
  codeForSubjectTree *
  gen_Inst_List_for_Subject_Tree(ast * p, int goalnt)
  ```

- If p is a sub-tree OP(leftOpd, rightOpd):
  - Generate instructions for the subtree corresponding to the goal non-terminal leftOpd
  - Generate instructions for the subtree corresponding to the goal non-terminal rightOpd
  - In which order should we generate the instructions?

Actions for Instruction Selection

- Task: Having identified a sub-tree, generate instructions for computing that sub-tree
- Recursive function

  ```
  codeForSubjectTree *
  gen_Inst_List_for_Subject_Tree(ast * p, int goalnt)
  ```

- If p is a sub-tree OP(leftOpd, rightOpd):
  - Generate instructions for the subtree corresponding to the goal non-terminal leftOpd
  - Generate instructions for the subtree corresponding to the goal non-terminal rightOpd
  - In which order should we generate the instructions?
  - Generate instruction for OP using the result of leftOpd and rightOpd

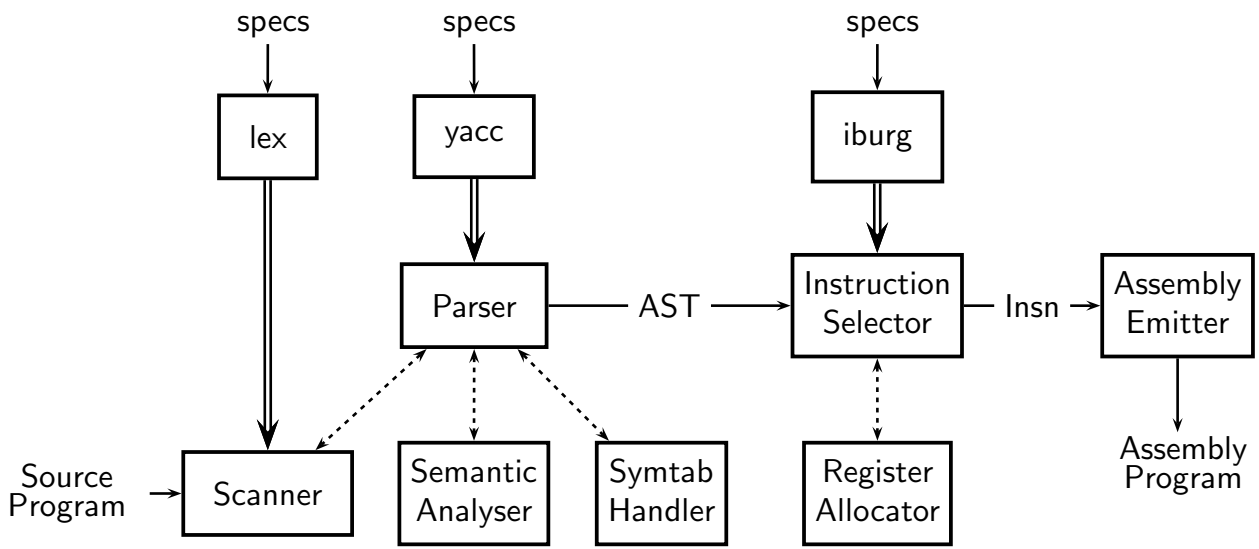
Actions for Instruction Selection

- Task: Having identified a sub-tree, generate instructions for computing that sub-tree
- Recursive function

  ```
  codeForSubjectTree *
  gen_Inst_List_for_Subject_Tree(ast * p, int goalnt)
  ```

- If p is a sub-tree OP(leftOpd, rightOpd):
  - Generate instructions for the subtree corresponding to the goal non-terminal leftOpd
  - Generate instructions for the subtree corresponding to the goal non-terminal rightOpd
  - In which order should we generate the instructions?
  - Generate instruction for OP using the result of leftOpd and rightOpd
  - Perform local register allocation

Putting It All Together




```
main(int argc, char * argv[])
{
        processOptions(argc, argv);
        sTree = get_Ast();
        insts = sub_Tree_List_To_Inst_List(sTree);
        inst_List_To_Assembly(insts);
}
```