

**Few official announcements:**

1. Project 4 is out
2. Midterm grades are out
3. Project 3 grades are out  
Contact Bernice if you have any questions  
<http://www.cs.purdue.edu/homes/jye/cs352/Apr-04.pdf>

**Back to our Project 4:**

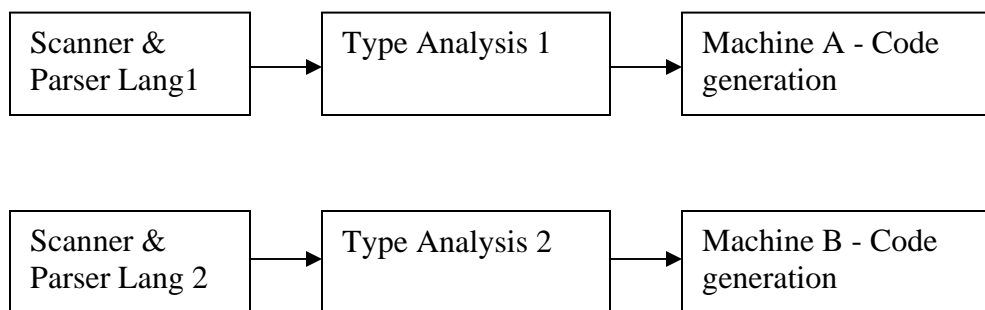
**1. Translation to Intermediate Code Trees**

- Read Chapter 7
- Another very good resource: Lecture slides available at:  
<http://www.cs.purdue.edu/homes/hosking/352/notes/06-trans.pdf>

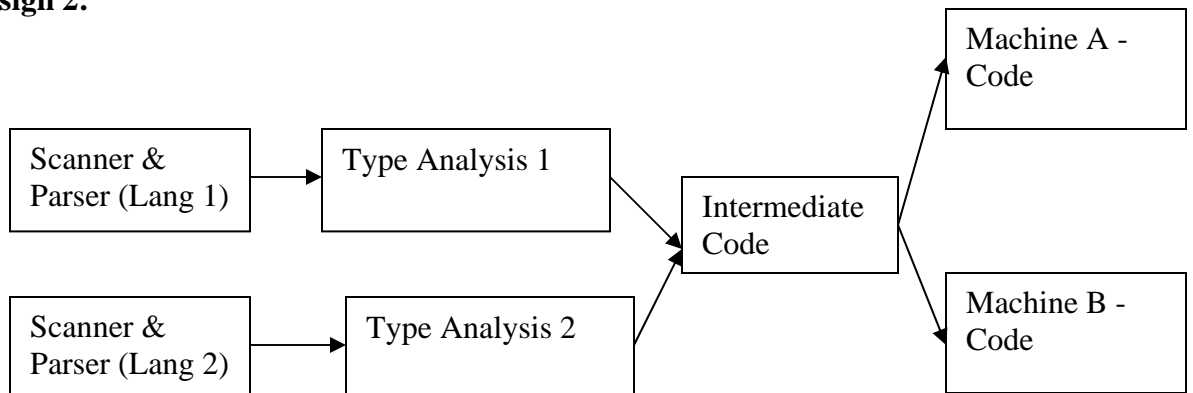
You might ask, why is the need for intermediate code, instead of direct machine code?

Lets see two different design methodologies for compiler design:

**Design 1:**



**Design 2:**



Design 2 is better, why?

- Separation of concerns
- Changes in Languages
- Changes in Machines

2. What you need in the Intermediate Tree:

Just like a normal machine code, not too specific and not too general. What are the advantages & disadvantages?

3. In our Intermediate code, we assume that infinite pool of registers are available.

4. Take a look at the `Tree.Exp` classes, `Tree.Stm` classes.

5. Each Method:

- Frame object
- `Tree.Stm`

Both are combinedly called as `procfrag`.

6. Starting Point:

1. Look at `Translate.java`

Visitors of `Absyn` classes. Returns `Translate.Exp` classes

2. Translate package classes

methods

`unEX`

`unNx`

`unCx`

returning `Tree.Exp` or `Tree.Stm`

Why all this mess? Consider the following:

We can't assume that `a=b` is a statement or an expression.

1. `If(a=b)`

`1;`

`else`

`2;`

2. `a=b;`

Depending on the context we decode the expression to `Exp`, `Stm` or `Conditional`

So we have three methods in `Translate.Exp` classes that do this:

`unEx`

`unNx`

`unCx`