# Compiler Design

Prof. Sankhadeep Chatterjee

Department of Computer Science & Engineering, UEMK

## Previous Class

- Front End Compilation
- Lexical Analysis
- Syntax Analysis
- Semantic Analysis

# Intermediate Code Generator

- Intermediate Code Generator coverts the unambiguous parse tree to an intermediate machine dependent representation.

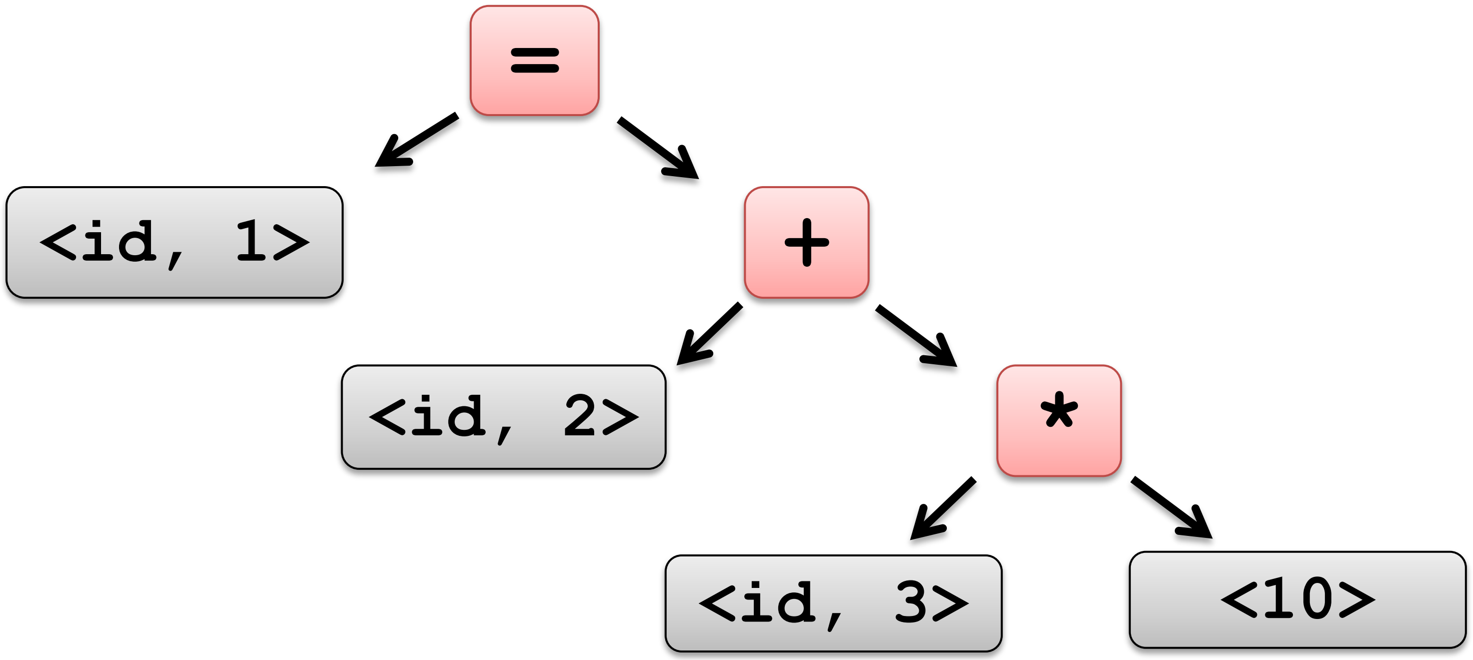
- There can be different ways of representations

- Three-address code is one such representation.

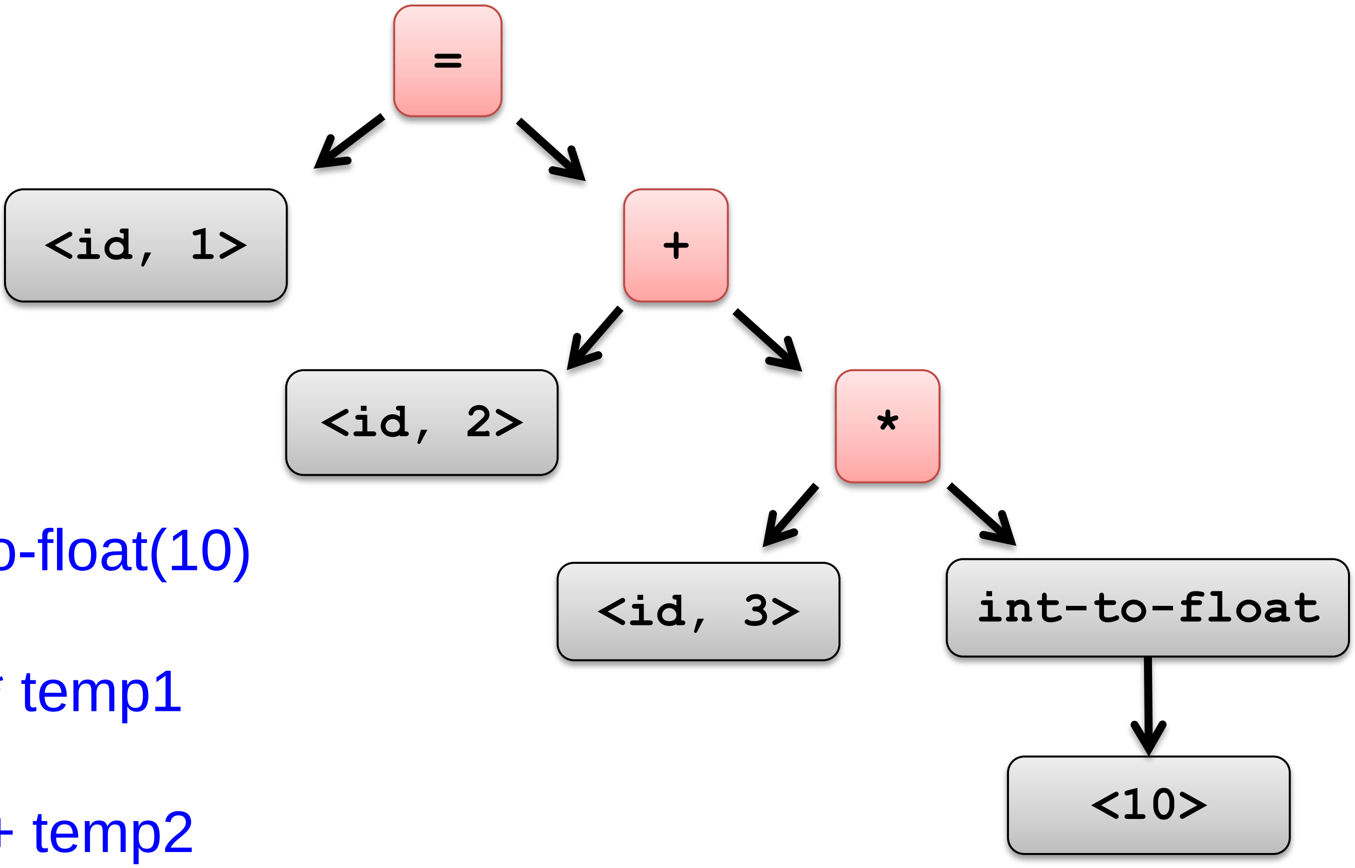
# Intermediate Code Generator

- Input character stream : `a = b + c * 10`
- Token Stream :

```
<id, 1> <=> <id, 2> <+> <id, 3> <*> <10>
```



# Intermediate Code Generator



temp1 := int-to-float(10)

temp2 := id3 * temp1

temp3 := id2 + temp2

id1 := temp3

# Code Optimization

- Automatically modify programs so that they
  - Run faster
  - Use less resources (memory, registers, space, fewer fetches etc.)
- Some common optimizations
  - Common sub-expression elimination
  - Dead code elimination , etc.

## Code Optimization

| **After Intermediate Code Generation** | **After Code Optimization** |
|---|---|
| temp1 := int-to-float(10) | temp1 := id3 * int-to-float(10) |
| temp2 := id3 * temp1 | id1 := temp1 + id2 |
| temp3 := id2 + temp2 | |
| id1 := temp3 | |

# Code Generation

- **Abstractions at the source level**

  identifiers, operators, expressions, statements, conditionals, iteration, functions (user defined, system defined or libraries)
- **Abstraction at the target level**

  memory locations, registers, stack, opcodes, addressing modes, system libraries, interface to the operating systems
- Code generation is mapping from source level abstractions to target machine abstractions

## Code Generation

- Map identifiers to locations (memory/storage allocation)
- Explicate variable accesses (change identifier reference to relocatable/absolute address
- Map source operators to opcodes or a sequence of opcodes

# Code Generation

- Convert conditionals and iterations to a test/jump or compare instructions
- Layout parameter passing protocols: locations for parameters, return values, layout of activations frame etc.
- Interface calls to library, runtime system, operating systems

# Code Generation

**Optimized Code**

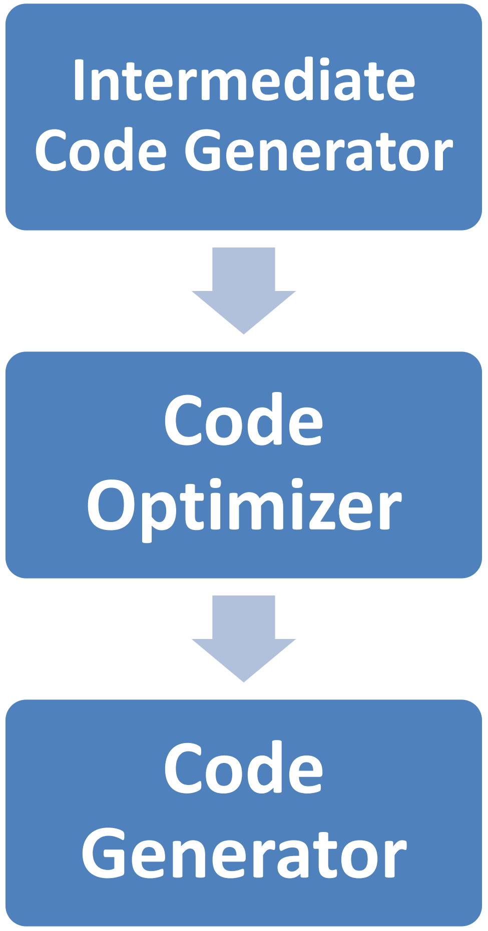
```
temp1 := id3 * int-to-float(10)

id1 := temp1 + id2
```

**Target Assembly Level Code**

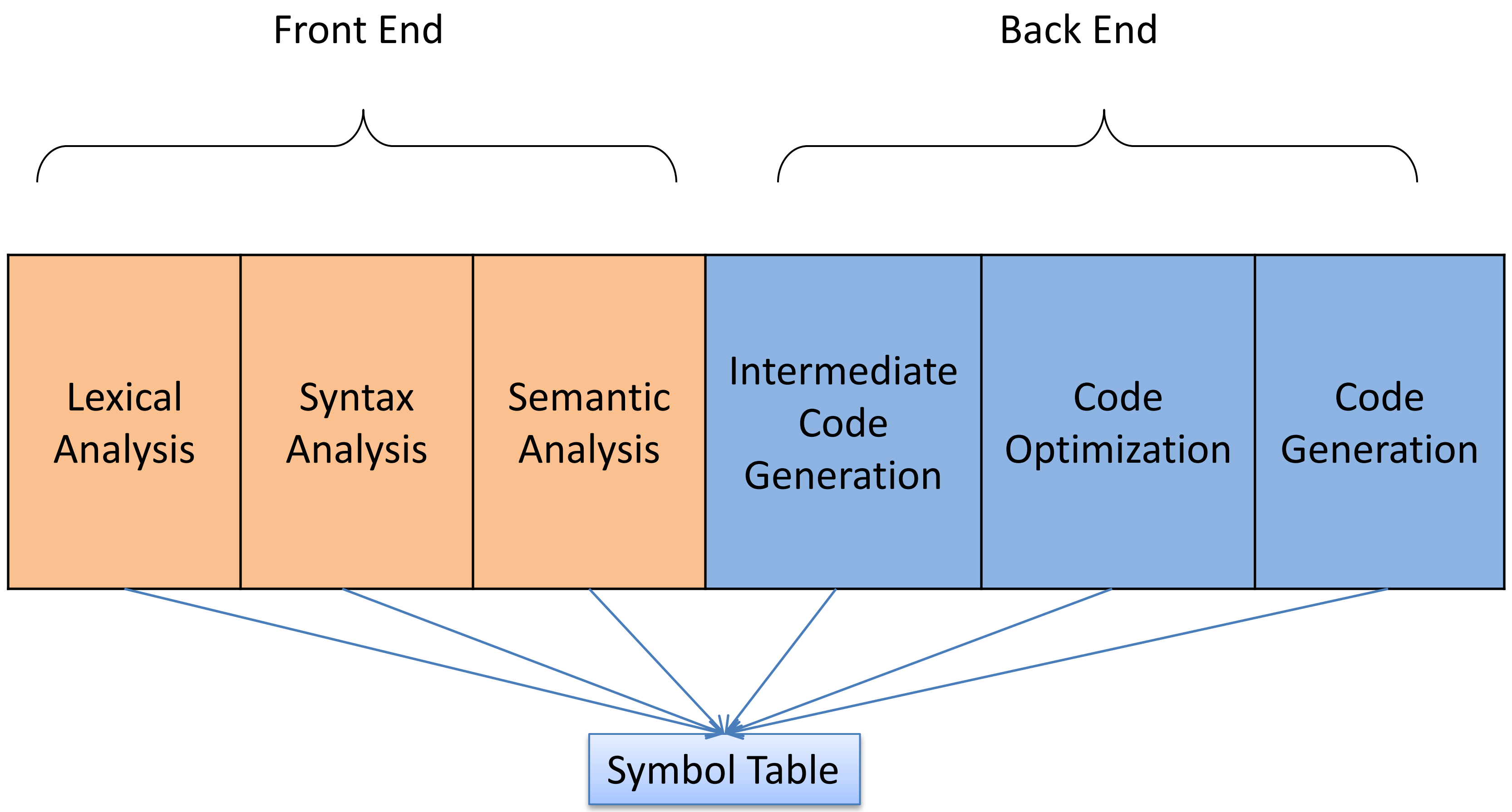
```
LDF    R2,   id3
MULF   R2,   R2,   #60.0
LDF    R1,   id2
ADDF   R1,   R1,   R2
STF    id1,  R1
```

# Compilation Phases

- Back End Compilation
- Machine Dependent



# Full Picture



## Thank You