Notes :

CENTURYPLY®

AH Sir

1 Lexical Analysis or Lexical Analyzer / Scanner
2 Syntax Analysis or Analyzer.
3 Semantic Analysis

All these are connected with Error Handler.

4 Intermediate Code Generation
5 Code Optimization
6 Code generation

All these are also connected with Symbol Table

Input is source code

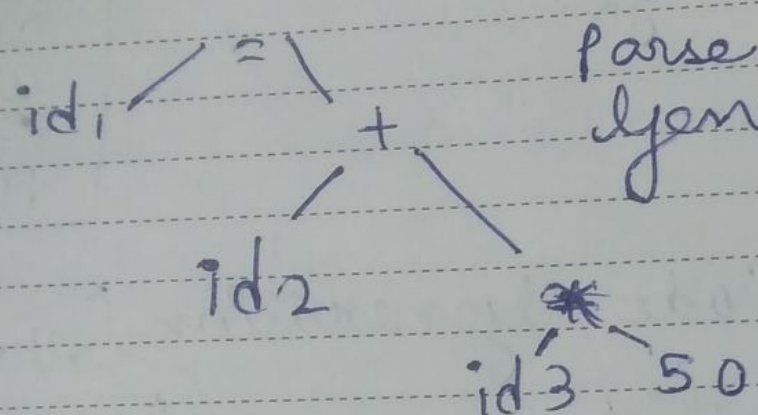
Total = number 1 + number * 50
id1 = id2 + id3 * 50

Token is sequence of character with collective meaning.

Delimeter

↓ After Syntax Analysis

Parse tree

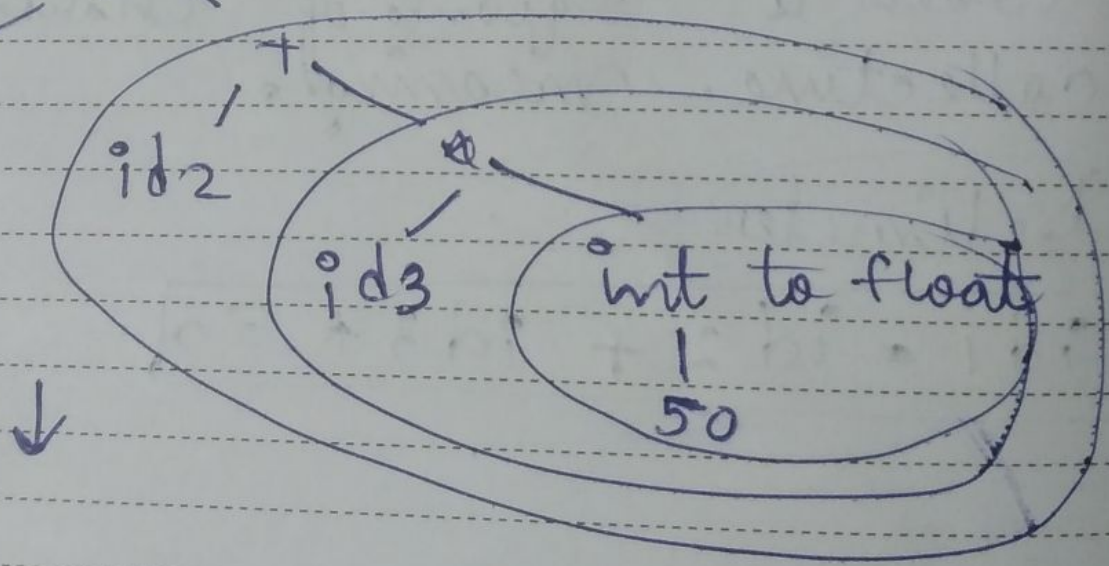


↓ Parse tree generation

(Now semantic → only for checking purpose)

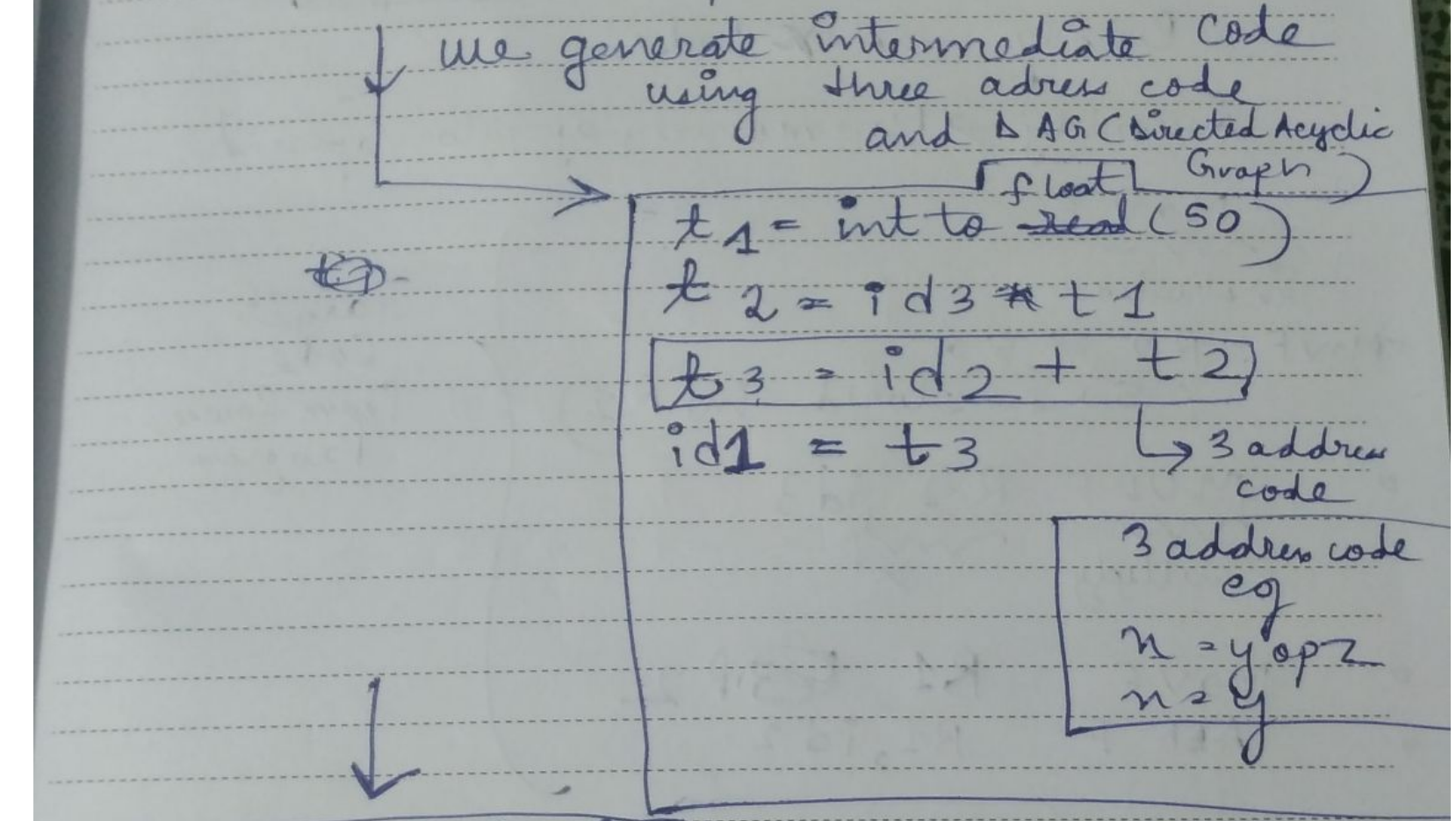
After semantic → output is parse tree but modified parse tree

↓ After semantic Analysis



↓

Front end → Depends on source code
Back end → Depends on intermediate code



Code Optimization

t3 = id3 * 50.0
id1 = id2 + t3

} Intermediate code

for 2 statements, time req is less.

Represent in Machine kernel language

↳ OS 1s

~~Binary ⇒ Hex Code~~

Assembly → Mnemonics, operator / operand

- MOVF
  - instruction
- MOVF R2, #50
  - (50 is stored in R2)
- MULF R2, id3
  - multiply
  - ×
- MOVF R1, R2
- ADD F R1, id2

Target Code from Source Program

CENTURYPLY®

example

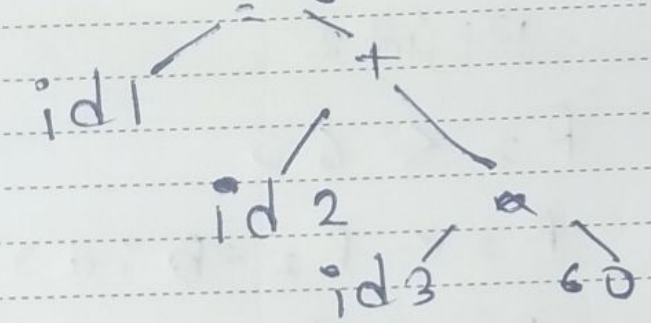
floating point

eg:- position = initial + rate * 60

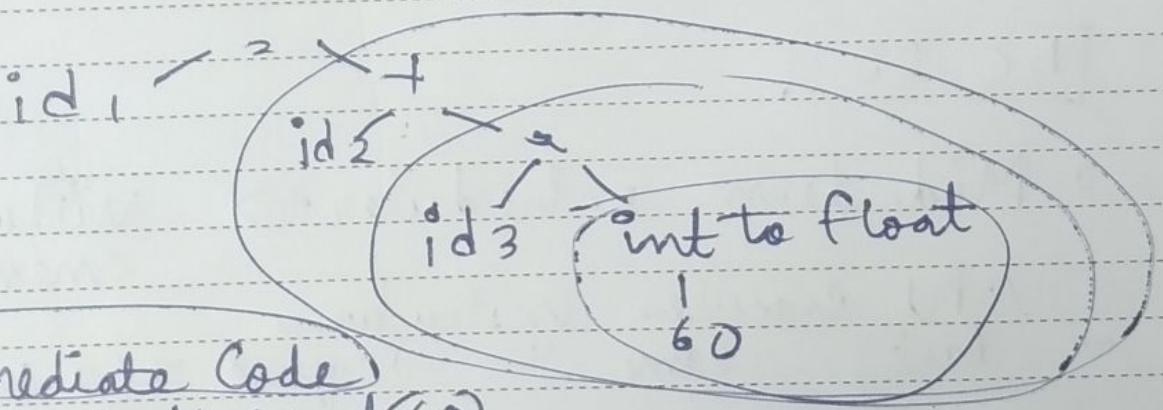
lexical Analyzer

id1 = id2 + id3 * 60

Syntax Analysis



Semantic Analysis



Intermediate Code

```
t1 = inttofloat(60)
t2 = id3 * t1
t3 = id2 + t2
id1 = t3
```

Code ~~co~~ optimization

```
t3 = id3 * 60.0
id1 = id2 + t3
```

Notes :

final assembly code

Code generation

- MOVF R2, #60
- MULF R2, id3
- MOVF R1, R2
- ADDF R1, id2

Just doing →

$R_2 \leftarrow 60$

$R_2 \leftarrow R_2 * id3$

$R_1 \leftarrow R_2$

$R_1 \rightarrow R_1 + id2$