Intermediate Code Generation

- Intermediate Code is generated using the Parse rule Producing a language from the input language.

Why do we need intermediate code?

-intermediate code has the following property – simple enough to be Translated to assembly code . complex enough to capture the complication of high level language.

Utility of Intermediate Code Generation:

1. Suppose We have n-source languages and m-Target languages. Without Intermediate code we will change each source language into target language directly.So,for each source-target pair we will need a compiler. Hence we will require (n*m) Compilers, one for each pair.

If we Use Intermediate code We will require n-Compilers to convert each source language into Intermediate code and m-Compilers to convert Intermediate code into m-target languages. Thus We require only (n+m) Compilers.

2. Retargetting is facilitated; a compiler for a different machine can be created by attaching a Back-end(which generate Target Code) for the new machine to an existing Front-end (which generate Intermediate Code).
3. A machine Independent Code-Optimizer can be applied to the Intermediate code.
4. Intermediate code is simple enough to be easily converted to any target code.
5. Complex enough to represent all the complex structure of high level language.

Types of Intermediate Code:

-High Level Representation.
-Low level Representation.

Code Optimization:

-See the total intermediate instruction and optimize.

-Find Redundancy in Code.
-find common subexpression
-check for dependency
-detect dead codes

TWO BASIC TYPES OF OPTIMIZATION:
1>machine dependent i.e utilizing hardware specification for max efficiency.but range of compatibility is small.
2>machine independent,

Syntax Tree is optimization of high level language.
Suppose we have following code:

```
if(x<0)  then   x=3*(y+1);
else   y=y+1;
```

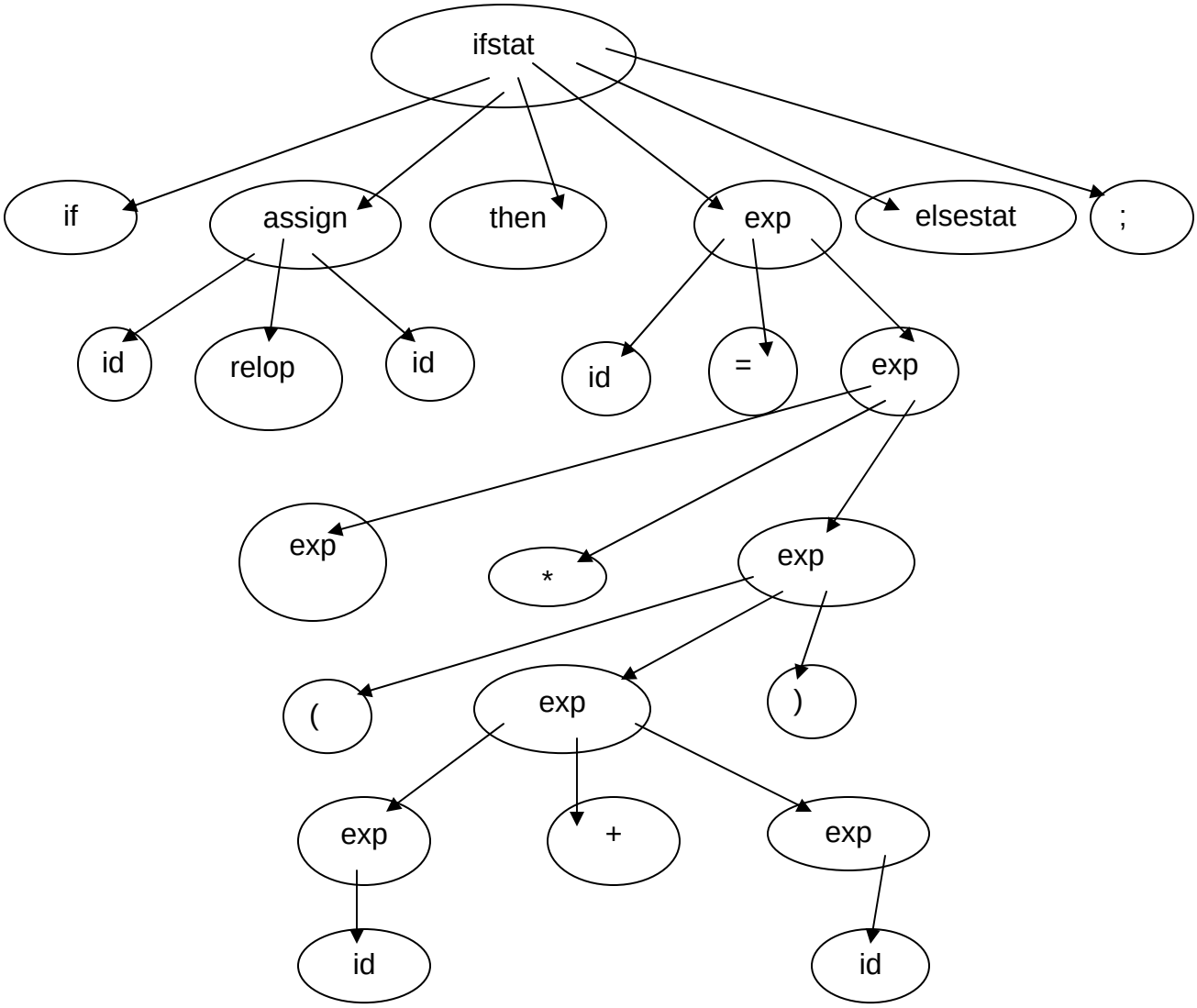
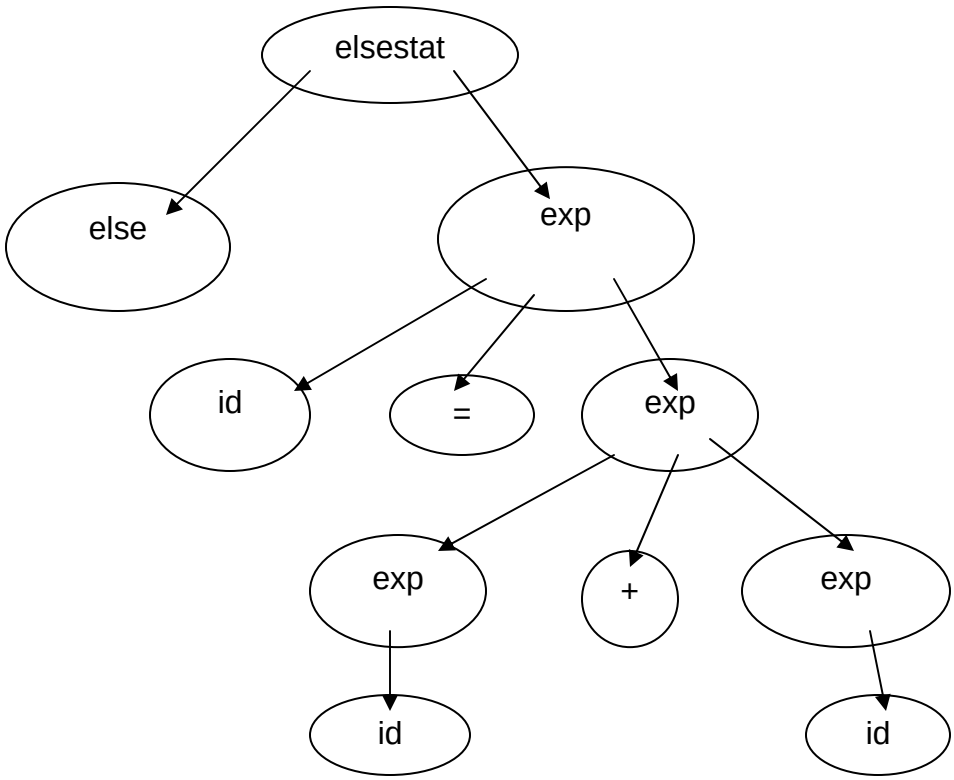


FIG → parse tree for ifstat(DAG).



We Can see A DAG gives the same information but in a more compact manner because common sub-expression are identified. PostFix Notation can be used to represent tree in linearized manner.In PostFix Notation edges of tree do not appear explicitly.

Three Address Code:

Another way to represent tree in linearized form is Three-address-code Notation.
In Three Address Code method internal nodes are given names.
It is a low level Intermediate code representation.

Example:

x=y *op* z

where x,y and z are names ,constants or Compiler generated temporaries.
where *op* stands for any operator.