CD : COMPILER DESIGN

Introduction to Compiler

Marwadi University

Department of CE

Unit no : 1
Introduction to Compiler
(01CE0601)

Prof. Shilpa Singhal

Outline :

Translator

Compiler, Interpreter, Assembler

Cousins of Compiler

Phases of Compiler

Grouping of Phases

Types of Compiler

Compiler construction tools

Pass and Phase

Marwadi University

Department of CE

Unit no : 1
Introduction to Compiler
(01CE0601)

Prof. Shilpa Singhal

Basic Terminology

Translator


BONUM MANE
Bonjour
καλημέρα
HELLO
guten morgen
dobro jutro
Labas rytas
Goeie môre
buenos días
sabah
good
BOM DIA


BONJOUR
HOLA
CIAO
HELLO

Translator

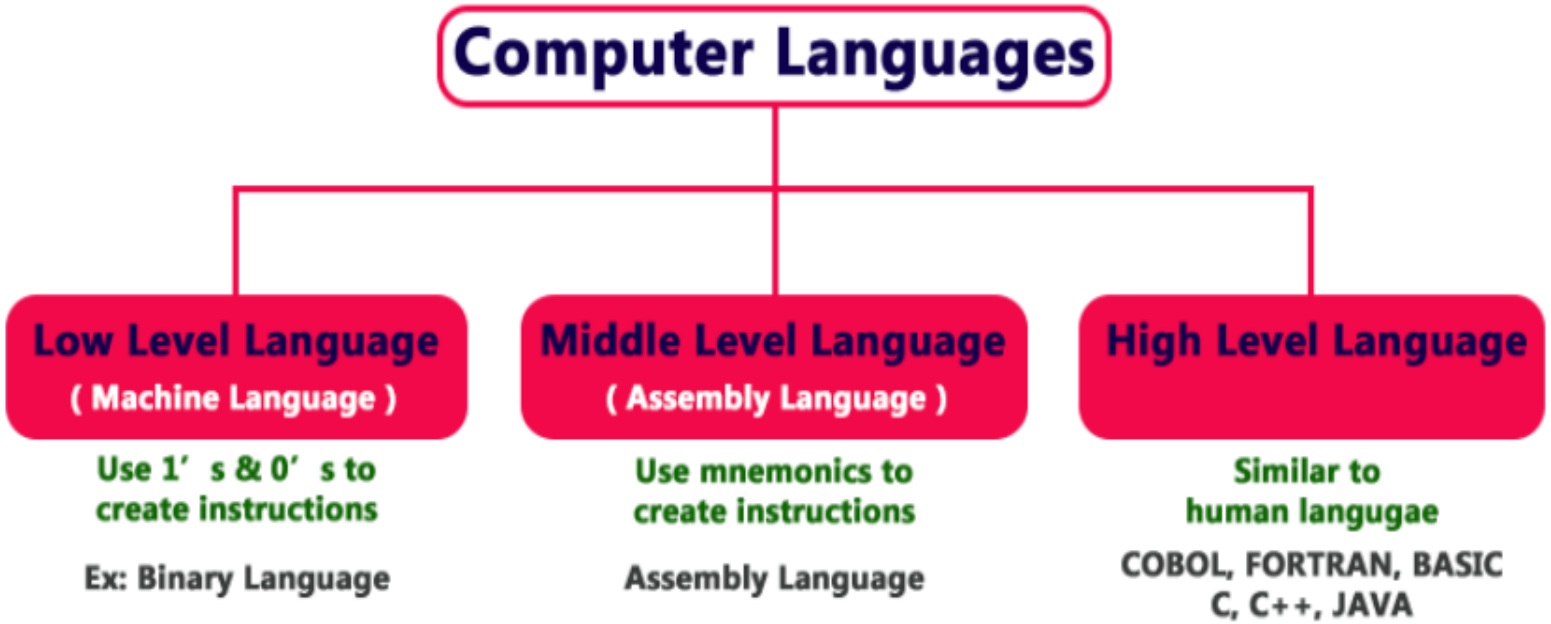
Translator :

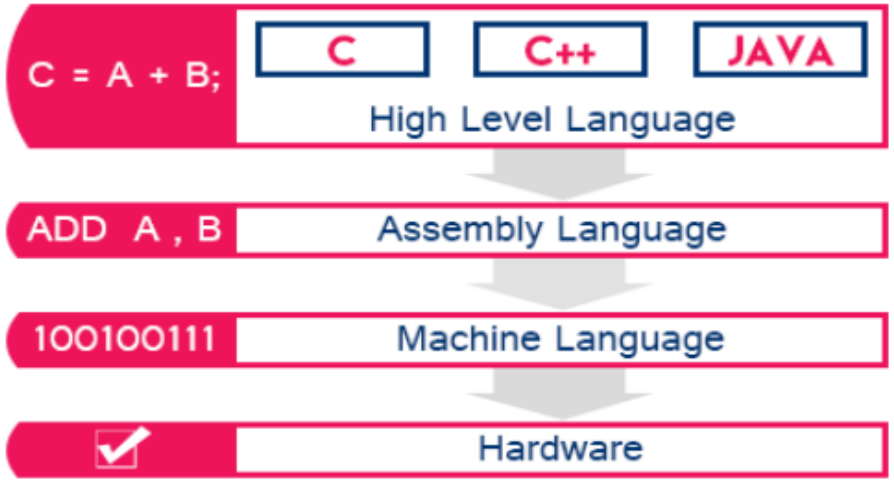
- A translator is a program which takes **one program** (written in one language) **as input** and **converts** it into **another language** (Target language).
- Input Language is Source Language and Output Language is Target Language.
- There are Three types of Translator :

1) Compiler
2) Interpreter
3) Assembler

- It also **detects and reports error** during translation.

Computer Languages





Computer Languages

- **High-level language** is a computer language which can be understood by the users. High-level language needs to be converted into the low-level language to make it understandable by the computer. We use **Compiler** or **interpreter** to convert high-level language to low-level language.

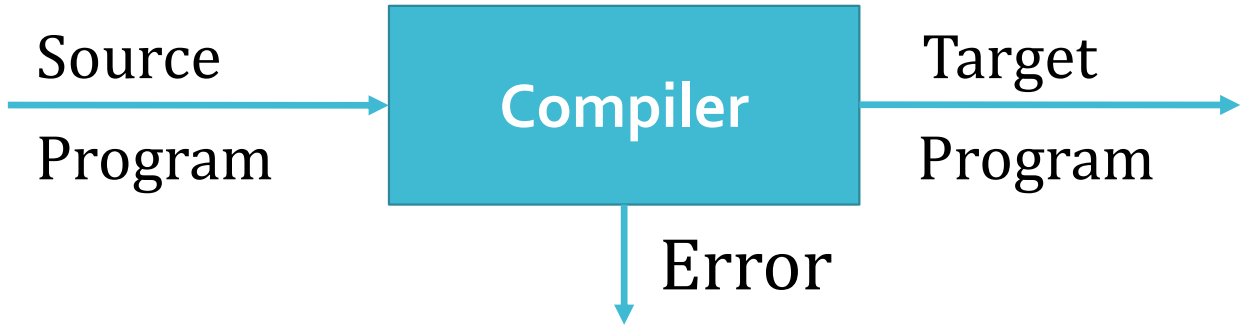
- **Middle-level language** is a computer language in which the instructions are created using symbols such as letters, digits and special characters. **Assembly language** is an example of middle-level language. In assembly language, we use predefined words called mnemonics. Assembler is a translator which takes assembly code as input and produces machine code as output.

- **Low-Level language** is the only language which can be understood by the computer. Low-level language is also known as **Machine Language**. The machine language contains only two symbols **1 & 0**.

1. Compiler

Compiler :

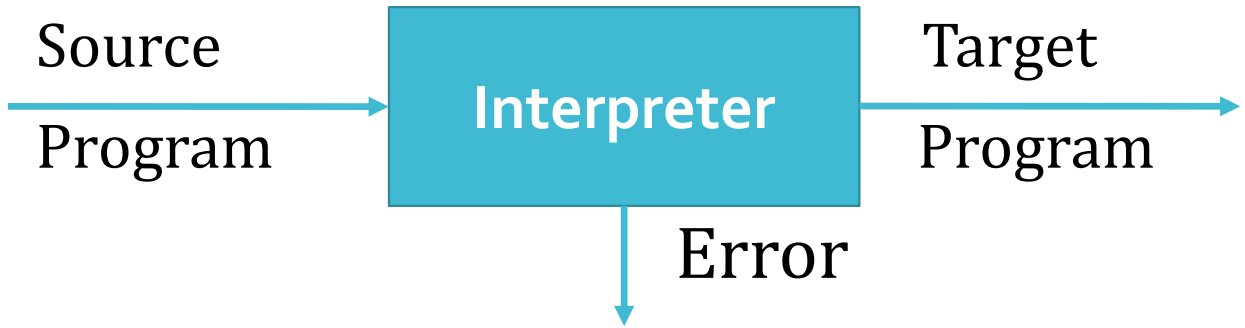
- Compiler is a program that reads a program written in one language that means source language and translates it into equivalent program into another language that is target language.
- Compiler is a program which translates **higher level language into functionally equivalent lower level language.**
- It also detects and reports error.



2. Interpreter

Interpreter :

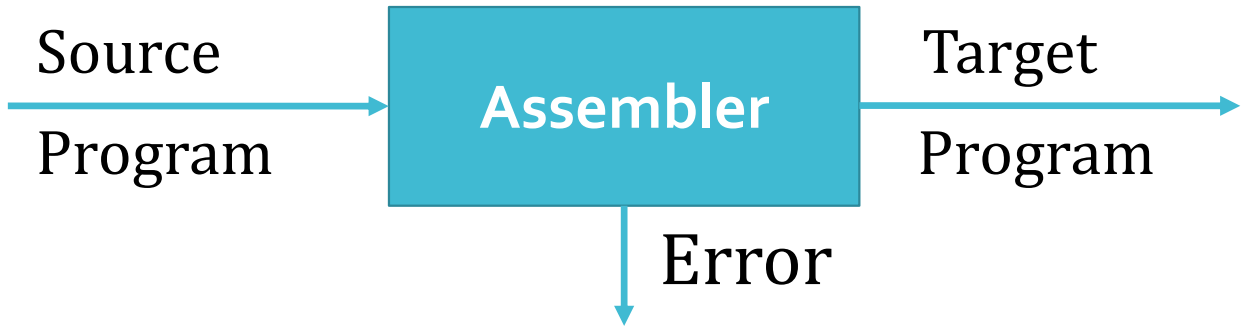
- Interpreter is a translator which is used to convert programs in high-level language to low-level language.
- Interpreter translates **line by line** and reports the error once it encountered during the translation process.



3. Assembler

Assembler :

- An assembler is a translator used to translate **assembly language to machine language.**
- An assembler translates a low-level language, an assembly language to an even lower-level language, which is the machine code. The machine code can be directly understood by the CPU.



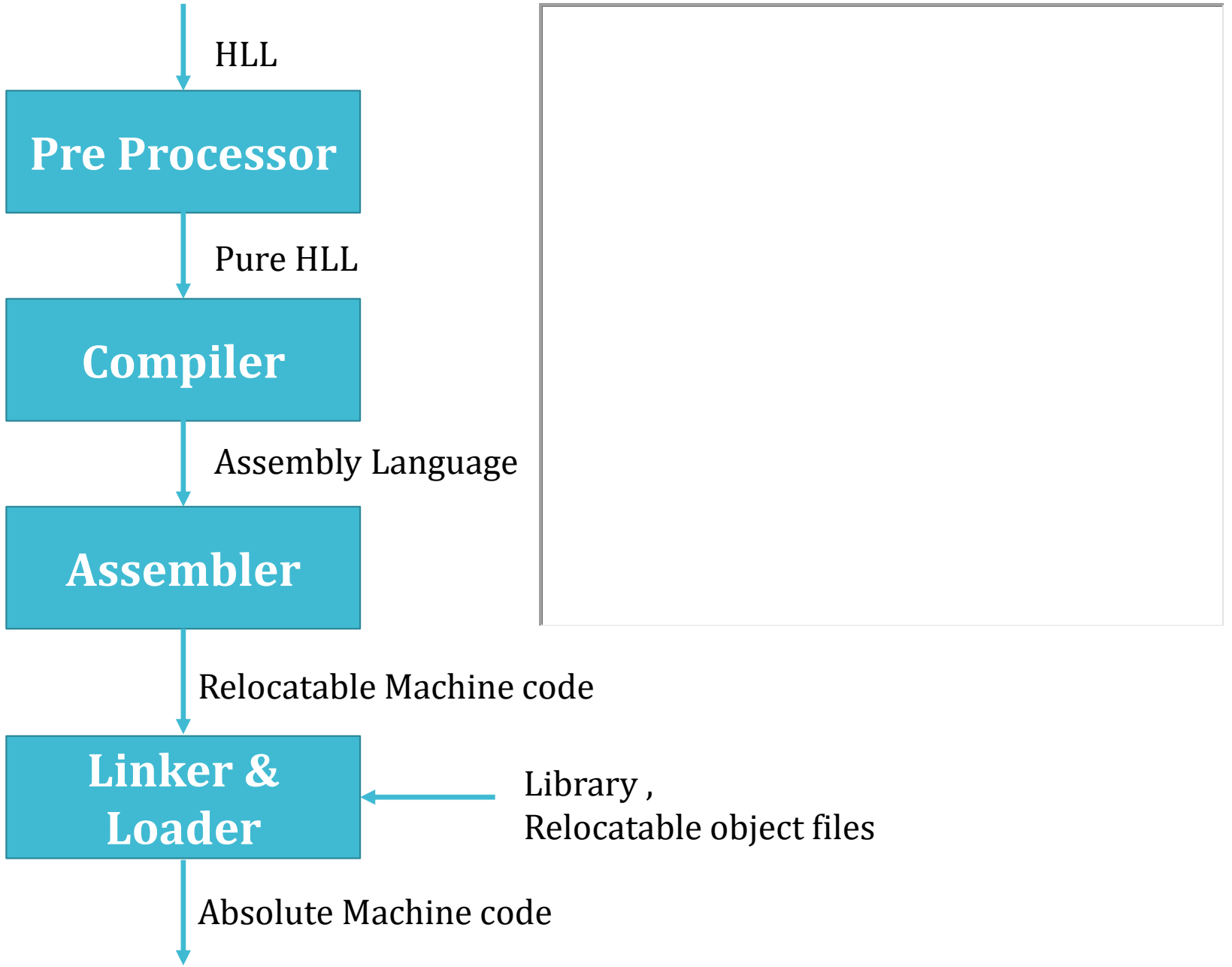
Compiler Vs Interpreter

Compiler	Interpreter
It takes whole program as input	It takes single instruction as input
It generates intermediate object code	It does not generate any intermediate object code
Memory requirement is more due to creation of object code.	Memory requirement is less (memory efficient) as it does not create intermediate object code.
It takes large amount of time to analyze the source code but overall execution is comparatively faster.	It takes less amount of time to analyze the source code but overall execution is slower.
Compilation is done before execution.	Compilation and execution take place simultaneously.
Display all errors after entire program is checked.	Display errors of each line one by one.
Ex : C , C ++	Ex : BASIC , Python

Compiler Vs Assembler

Compiler	Assembler
Input for compiler is Pre processed Source code.	Input for Assembler is Assembly language code.
It generates assembly language code or directly the executable code.	It generates the relocatable machine code.
There are 6 Phases of compiler	It makes Two passes over given input.
Ex : C , C ++	Ex : 8085 , 8086

Language Processing System / Translation Processor / Context of Compiler /Cousins of Compiler



Language Processing System / Translation Processor / Context of Compiler /Cousins of Compiler

- There are three cousins of Compiler

1) Preprocessor
2) Assembler
3) Linker and Loader

Language Processing System / Translation Processor / Context of Compiler /Cousins of Compiler

1) Preprocessor

- Preprocessor produces input to compilers.
- They may perform the following functions

➢ **Macro Processing** : A preprocessor may allow a user to define macros that are short hands for longer constructs.

➢ **File Inclusion** : A preprocessor may include header files into program text.

➢ **"Rational" Preprocessor** : These processors augment older languages with more modern flow of control and data structuring facilities. (Built in Macro for construct like While or If Statement)

➢ **Language Extension** : These processor attempt to add capabilities to the language by what amount to built in macros. the language equal is a database query language embedded in C. statement beginning with ## are taken by preprocessor to be database access statement unrelated to C and translated into procedure call on routines that perform database access.

Language Processing System /Translation Processor / Context of Compiler /Cousins of Compiler

1) Preprocessor

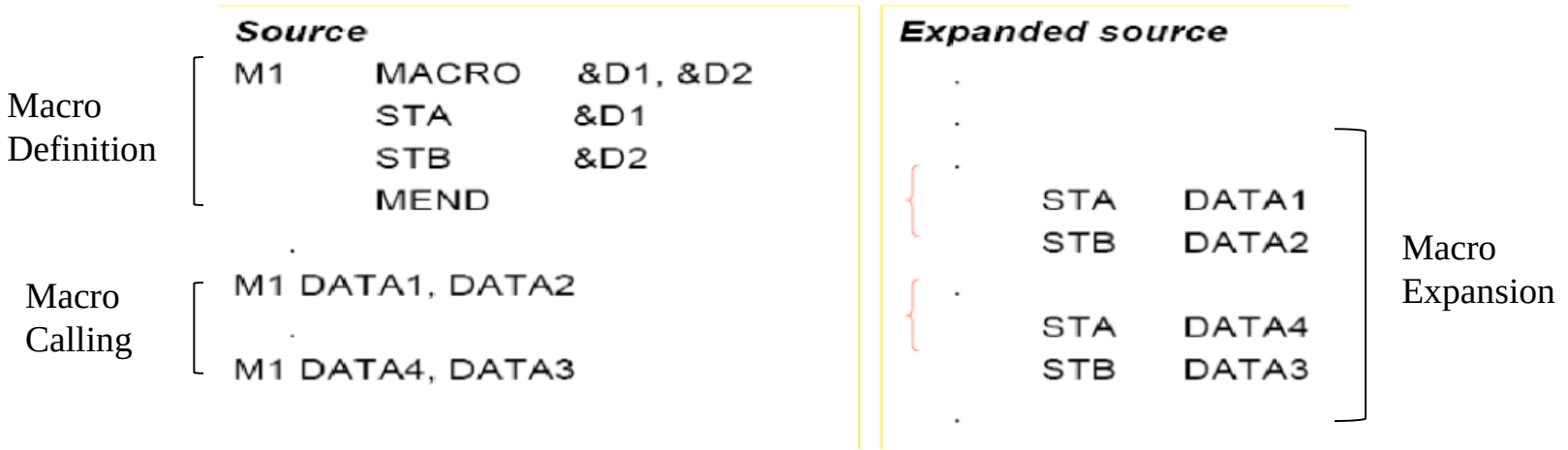
Macro processor deal with following statements:

→ Macro Definition:

MACRO: identify the beginning of a macro definition

MEND: identify the end of a macro definition

→ Macro Calls and expansion:



M1 is a macro with two parameters D1 and D2. The MACRO stores the contents of register A in D1 and the contents of register B in D2. Later M1 is invoked with the parameters DATA1 and DATA2

Language Processing System /Translation Processor / Context of Compiler /Cousins of Compiler

The macro named SUM is used to find the sum of two variables passed to it.

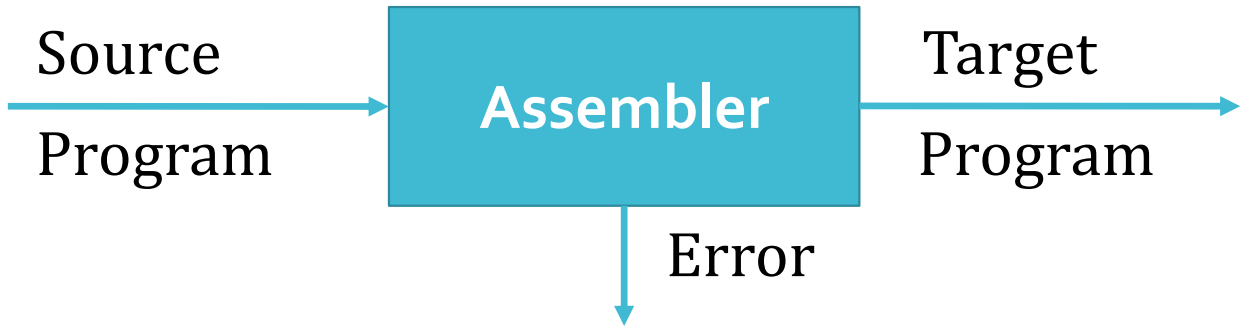
```
SUM     MACRO   &X,&Y
        LDA     &X
        MOV     B
        LDA     &Y
        ADD     B
        MEND
```

- MCRP-Start Macro
- LDA-Load the value of &X into the accumulator.
- Move the value from the accumulator to register B.
- LDA-Load the value of &Y into the accumulator.
- Add the value in register B (which is &X) to the value in the accumulator (which is &Y).The result of &X + &Y is now in the accumulator.
- MEND-End Macro

Language Processing System / Translation Processor / Context of Compiler /Cousins of Compiler

2) Assembler

- An assembler is a translator used to translate **assembly language to machine language.**
- An assembler translates a low-level language, an assembly language to an even lower-level language, which is the machine code. The machine code can be directly understood by the CPU.



Language Processing System / Translation Processor / Context of Compiler /Cousins of Compiler

Assembly code is mnemonic version of machine code, in which names are used instead of binary codes for operation and names are also given to memory address.

Example:

```
MOV     AX, X
```

--MOV is a mnemonic opcode.

--AX is a register operand in symbolic form.

--X is a memory operand in symbolic form.

Language Processing System / Translation Processor / Context of Compiler /Cousins of Compiler

2) Assembler

Instruction Opcode	Assembly Mnemonic	Remarks
00	STOP	Stop Execution
01	ADD	Op1 ← Op1+ Op2
02	SUB	Op1 ← Op1 – Op2
03	MULT	Op1 ← Op1* Op2
04	MOVER	CPU Reg ← Memory operand
05	MOVEM	Memory ← CPU Reg
06	COMP	Sets Condition Code
07	BC	Branch on Condition
08	DIV	Op1 ← Op1/ Op2
09	READ	Operand 2 ← input Value
10	PRINT	Output ← Operand2

Fig: Mnemonic Operation Codes

Language Processing System / Translation Processor / Context of Compiler /Cousins of Compiler

2) Assembler

```
        START   101
        READ    X
        READ    Y
        MOVER   AREG, X
        ADD     AREG, Y
        MOVEM   AREG, RESULT
        PRINT   RESULT
        STOP
X       DS      1
Y       DS      1
RESULT  DS      1
        END
```

Fig: Sample program to find X+Y

- DS // Declare Storage = 1 'word' for X,Y,RESULT
- DC // Declare Constant

Language Processing System / Translation Processor / Context of Compiler /Cousins of Compiler

3) **Linker and Loader**

- A program called loader performs the two functions of loading and link editing
- The process of loading consist of taking relocatable machine code , altering the relocatable address and placing the altered instructions and data in memory at the proper location.
- Linker allows us to make a single program from several files of relocatable machine code.
- Types of Linking : Static and Dynamic
- Types of Loader : Compile and Go loader, Absolute Loader, Relocating Loader.

Phases of Compiler

Lexical Analyzer

Phases of Compiler / Analysis – Synthesis Model of Compilation

1) Lexical Analysis

- It is also called **Linear analysis** or **scanning.**
- In this phase, input character from **source code is read** from left to right and then **break into stream of units.** These units are called **tokens.**
- Sequence of characters having a collective meaning is called Token.
- Tokens can be categorized into identifiers, constants, literals, keywords, operators, delimeters etc.
- So, Token is the smallest meaningful entities of program are produced as output.

Phases of Compiler / Analysis – Synthesis Model of Compilation

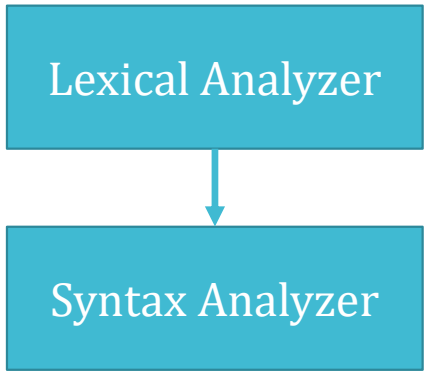
1) Lexical Analysis

Example : position = initial + rate * 60

id1 = id2 + id3 * 60

Position	Identifier 1
=	Equal to operator
Initial	Identifier 2
+	Addition Operator
Rate	Identifier 3
*	Multiplication Operator
60	Constant

Phases of Compiler



Phases of Compiler / Analysis – Synthesis Model of Compilation

2) Syntax Analysis

- It is also called **Hierarchical analysis** or **Parsing.**
- It determines if the sentence formed from the words are syntactically (grammatically) correct.
- It creates **syntax tree** from generated tokens if the code is error free.
- Syntax tree consist operators as internal node and operands as leaf node.
- This phase check each and every line and try to detect errors if it is grammatically (syntax wise) not correct.

Phases of Compiler / Analysis – Synthesis Model of Compilation

2) Syntax Analysis

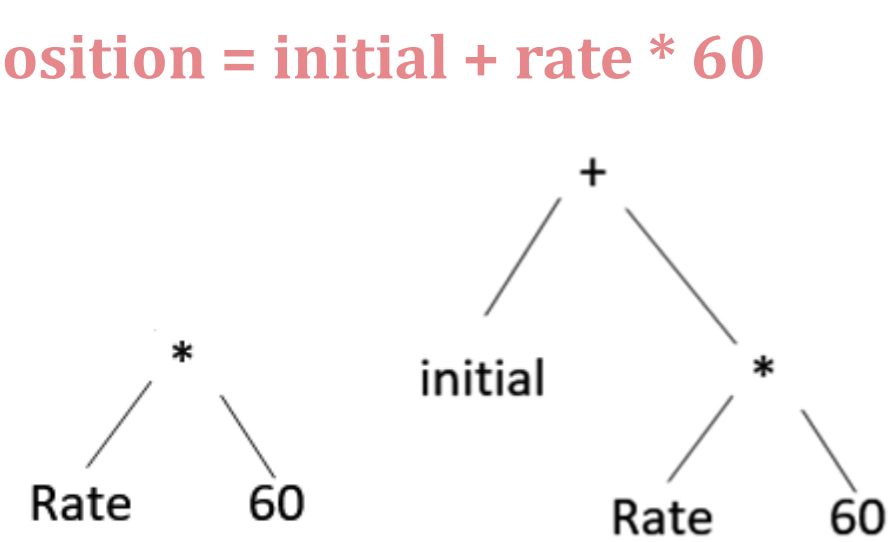
position = initial + rate * 60

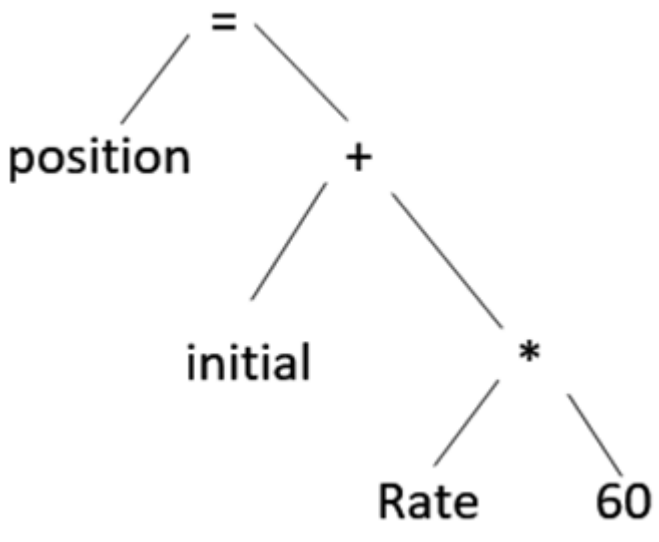
Three Operators : = + *

position = initial + **rate * 60**

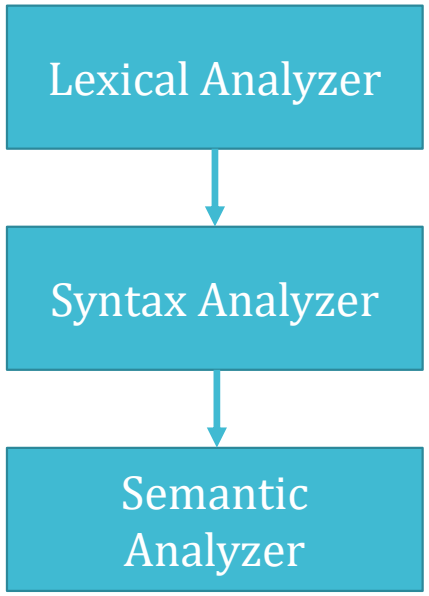
position = **initial + rate * 60**

position = initial + rate * 60





Phases of Compiler



Phases of Compiler / Analysis – Synthesis Model of Compilation

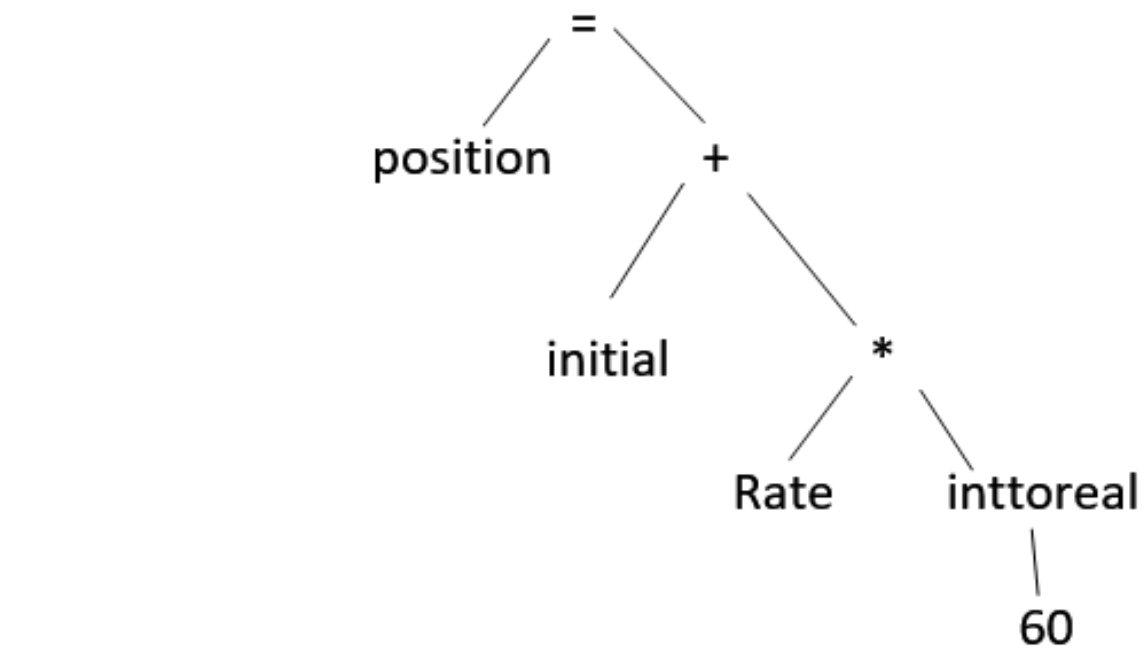
3) Semantic Analysis

- It **determines meaning** of string
- In semantic analysis various operations are performed like
- Performing check ; whether operator have compatible arguments or not (type checking) , matching of parenthesis, scope of operation etc.
- Ensuring that components of a program fits together meaningfully.

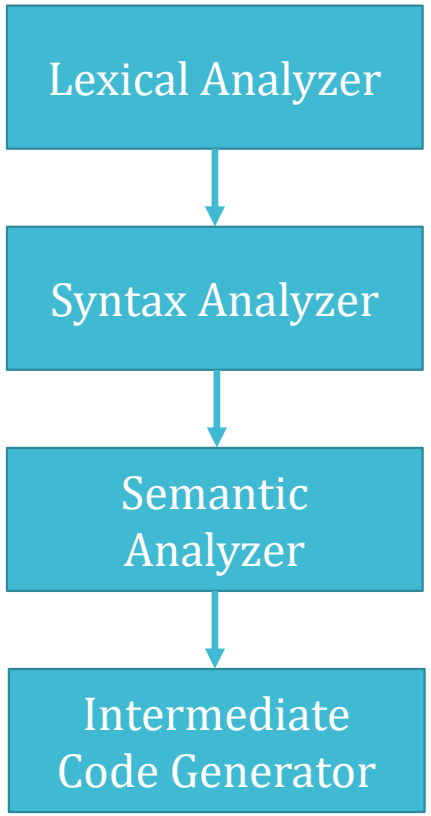
Phases of Compiler / Analysis – Synthesis Model of Compilation

3) Semantic Analysis

position = initial + rate * 60



Phases of Compiler



Phases of Compiler / Analysis – Synthesis Model of Compilation

4) Intermediate Code Generator

- Intermediate code generation should have two properties :

  it should be easy to produce,

  easy to translate.

- Intermediate code is called "**Three Address Code**".
- It is called three address code as it maximum consist three operands.

Phases of Compiler / Analysis – Synthesis Model of Compilation

4) Intermediate Code Generator

Example : position = initial + rate * 60

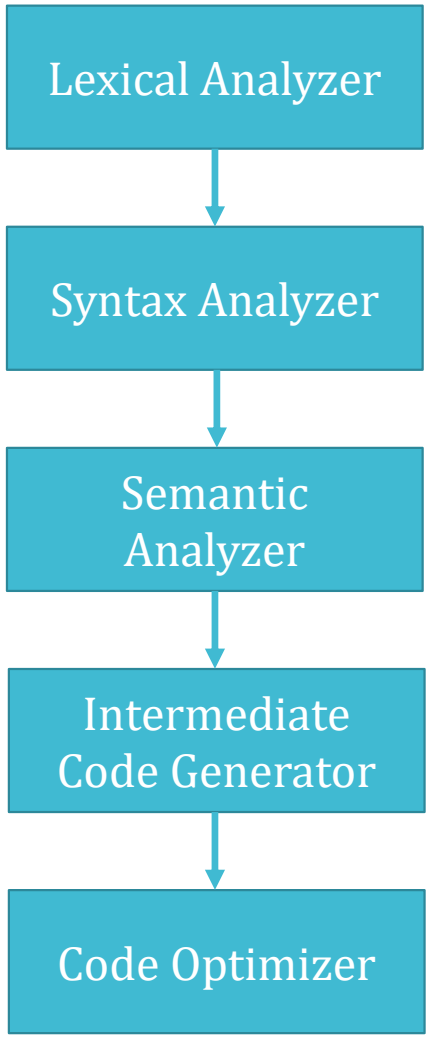
t1 = inttoreal(60)

t2 = rate * t1

t3 = initial + t2

position = t3

Phases of Compiler



Phases of Compiler / Analysis – Synthesis Model of Compilation

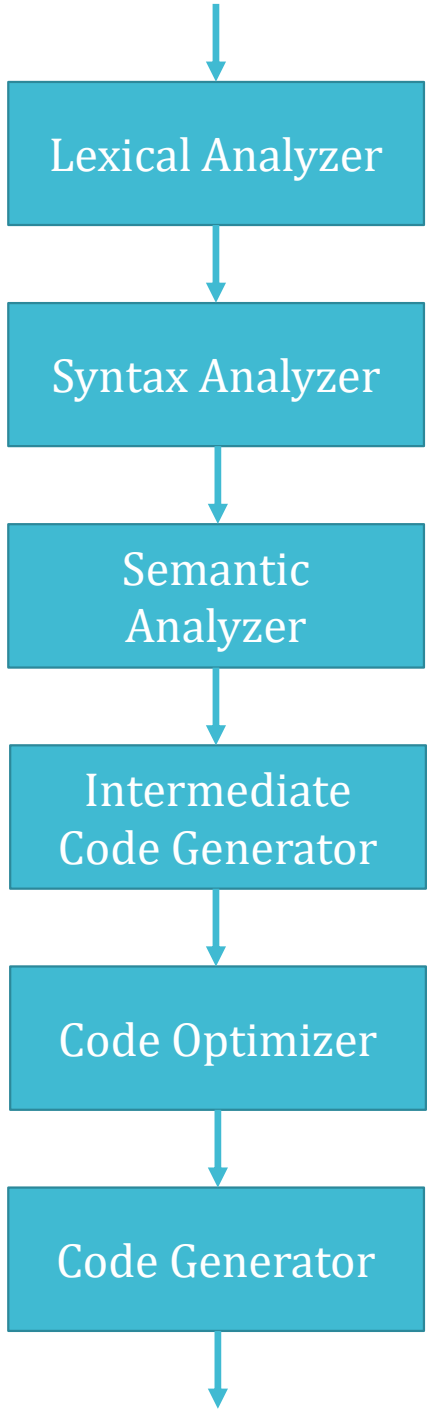
5) Code Optimization

- This phase improves the intermediate code, in such a way that a machine code can be produced, which occupies **less memory space** and **less execution time** without changing the functionality or correctness of program.
- Example :

```
t1 = rate * 60.00

position = initial + t1
```

Phases of Compiler



Phases of Compiler / Analysis – Synthesis Model of Compilation

6) Code Generation

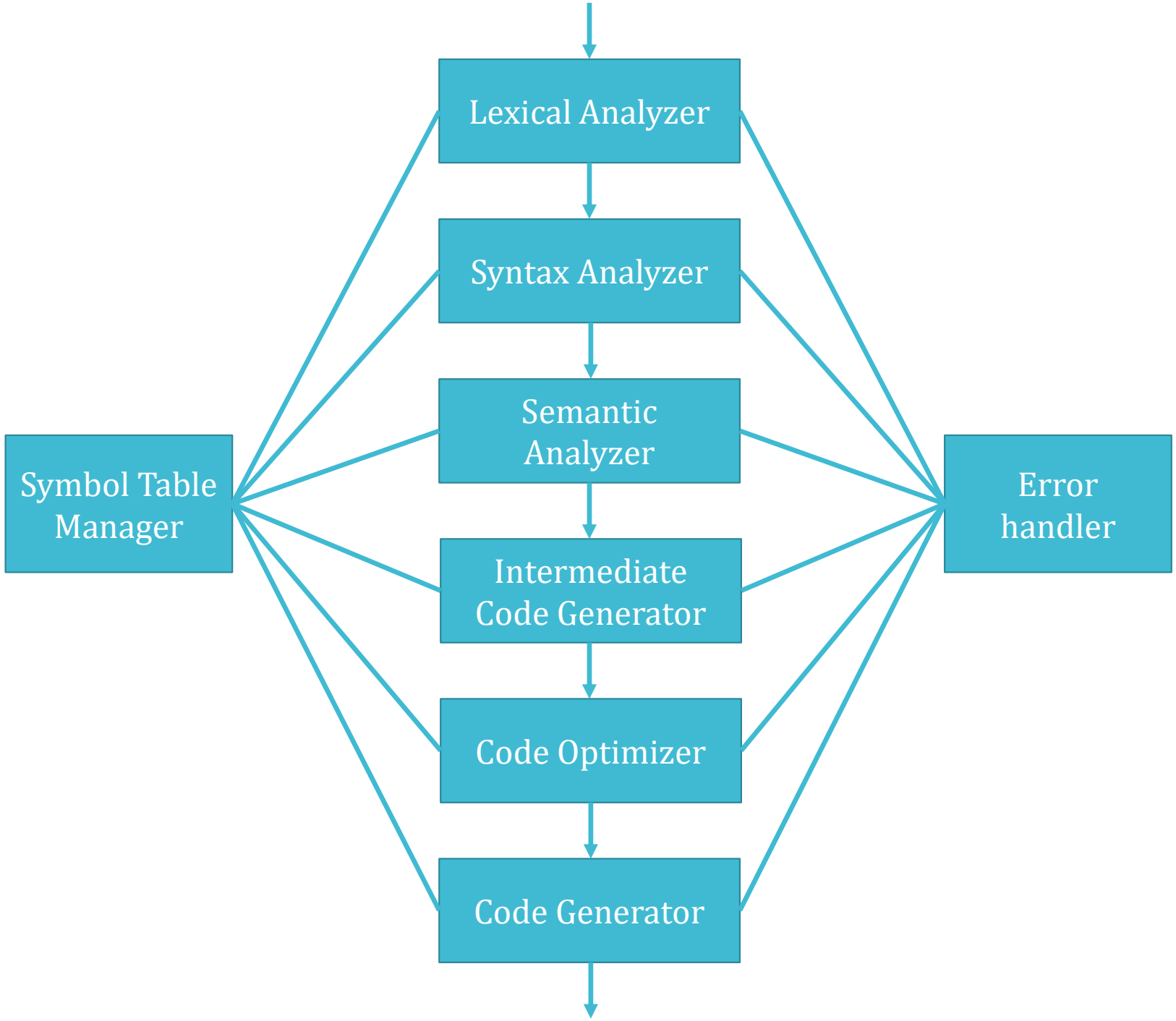
- In Code generation phase the target code gets generated.
- In this phase intermediate (optimized) code is translated into a sequence of machine instructions that perform the same operation.
- Example :

```
MOVF  id3,R1
MULF  #60.00, R1
MOVF   id2, R2
ADDF  R2, R1
MOVF  R1, id1
```

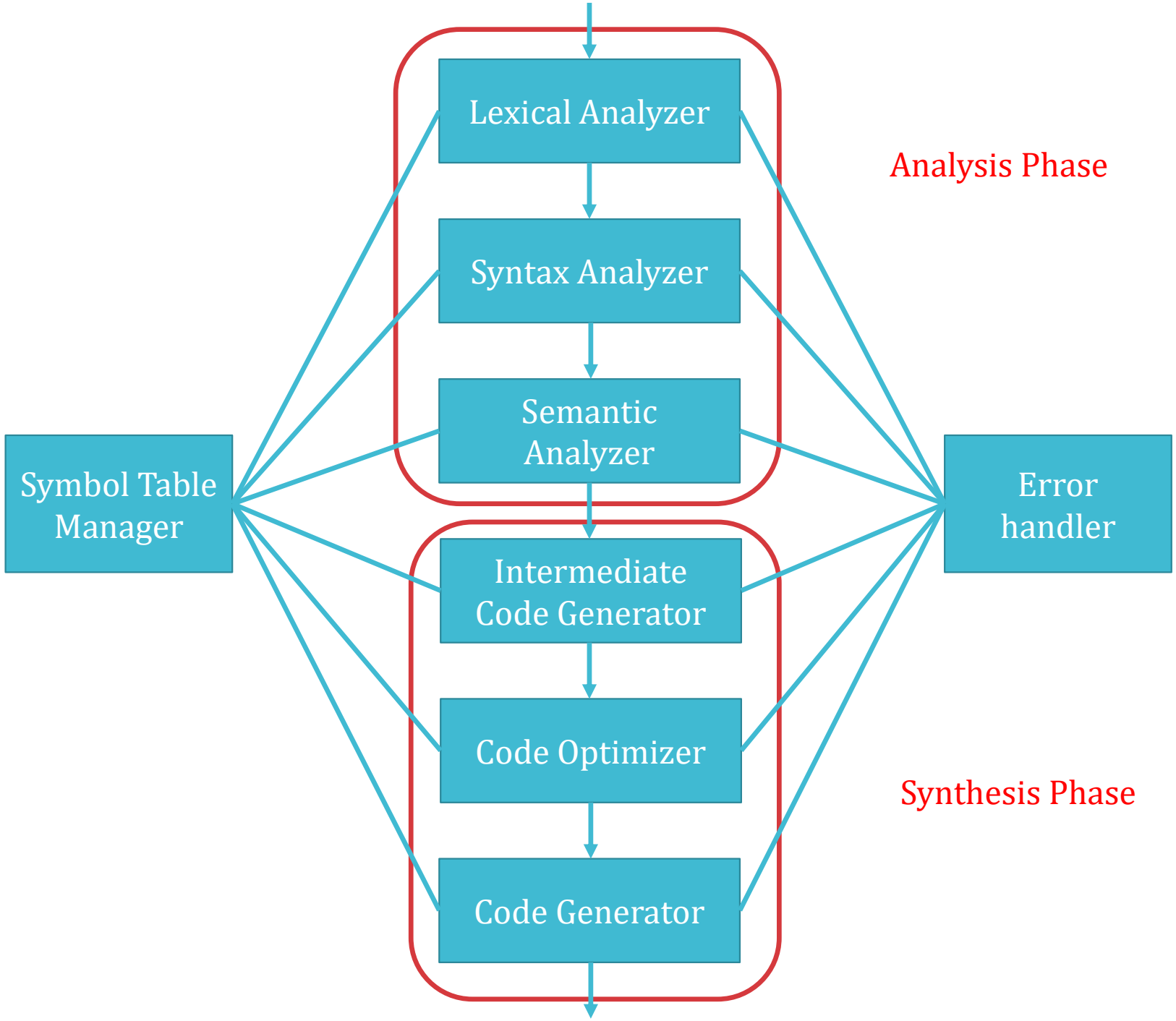
```
position = initial + rate * 60
    id 1        id2       id3

R1 = id3 (rate)
R1 = 60.00 * R1
R2 = id2 (initial)
R1 = R2 + R1
id1(position) = R1
```

Phases of Compiler



Phases of Compiler



Phases of Compiler / Analysis – Synthesis Model of Compilation

- There are two parts of Compilation Process :

1) Analysis Phase

   It breaks up source program into small pieces and create an intermediate representation of source program.

2) Synthesis Phase

   It constructs the desired target program from intermediate representation.

Phases of Compiler / Analysis – Synthesis Model of Compilation

1) Analysis Phase

Consist of Three Sub Phases :

- Lexical analysis
- Syntax analysis
- Semantic analysis

2) Synthesis Phase :

Consist of Three Sub Phases :

- Intermediate Code generation
- Code Optimization
- Code Generation

Phases of Compiler / Analysis – Synthesis Model of Compilation

Symbol Table :

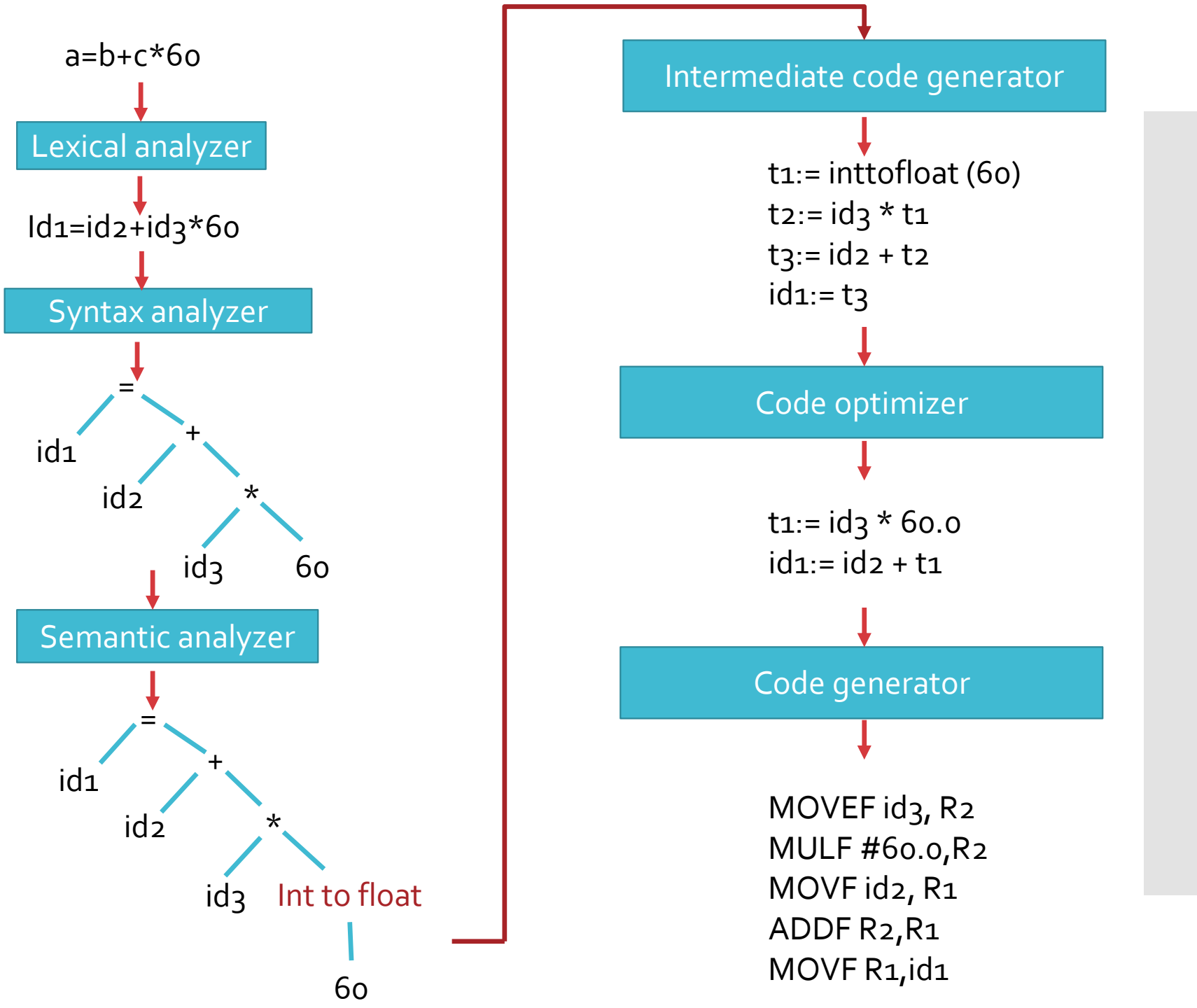
- A symbol table is **data structure** used by language translators.
- In order to keep track of identifiers used by program, compiler maintains the record of the same with their various attributes.
- Symbol table stores information for different entities : Identifier, Function, Procedure, Constant , Label Name, Compiler generated temporaries etc.
- All phases of compiler interact with symbol table Manager.

Phases of Compiler / Analysis – Synthesis Model of Compilation

Error Detection and Recovery :

- Each phase can encounter errors, However after detecting an error a phase must some how deal with that error. So that Compilation can proceed, allowing further errors in the source program to be detected.
- Lexical Analyzer can detect errors where the character remaining in the input do not form any token of the language.
- Errors where the token stream violates the structure rule (syntax) of the language are determined by syntax analysis phase.
- During Semantic analysis, the compiler tries to detect constructs that have the right syntactic structure but no meaning to the operation involved.

Phases of Compiler / Analysis – Synthesis Model of Compilation



Practice Time

Example :

What will be the output of each phase for the following statement?

Statement : x = a + b * c

Grouping of Phases

Front End

- **Dependent :** Source Language
- **Independent :** Target Machine

Back End

- **Dependent :** Target Machine
- **Independent :** Source Language

Grouping of Phases

Front End:

- Front end consist of those phase, that depends preliminary on the source language & are independent of the target machine.
- It includes **Lexical analysis, Syntax analysis, Semantic analysis, Intermediate code generation and Creation** of Symbol table.
- A certain amount of code optimization can be done by the front end as well.
- The front end also includes error handling that goes along with each of these phases.

Grouping of Phases

Back End:

- Back includes those portion of compiler that depend on the target machine and do not depend on source language.
- Back end includes **code optimization** and **code generation** with necessary error handling and symbol table operation.
- It has become fairly routine to take front end of a compiler & redo its back end to produce a compiler for the same source language on a different machine.

Pass of Compiler

- One Complete scan of Source program is called **Pass**.
- Difficulty in single pass :

→ **Forward Reference** : A forward reference means variable or label is referenced before its declared.

- It leads to Multi-Pass model.

Pass – 1 : Perform analysis of source program and note relevant information

Pass – 2 :Generate target code using information noted in pass-1.

```
          START    100
          READ      A
LOOP      MOVER    AREG  ,  A
             ........
             ........
          BC       GT , LOOP
          STOP


A         DS        1
```

Pass Vs Phase

Pass	Phase
Various phases are logically grouped together to form a pass.	The process of compilation is carried out in various step is called Phase.
Require Less Memory	Require More Memory
Execution is faster	Execution is slower
Less memory operation(read/write) to be performed	More memory operations to be performed
2 Passes of Compiler	6 Phases of Compiler

Types of Compiler

- Native Compiler
- Cross Compiler
- Source to Source Compiler
- Just in Time Compiler
- Incremental Compiler
- Parallelizing Compiler

Types of Compiler

1. Native Compiler :

- The compiler used to Compile a source code for **same type of platform only.**
- The output generated by this type of compiler can only be run on the same type of computer system and OS that the compiler itself runs on. For example, Turbo C or GCC compiler.

2. Cross Compiler :

- The compiler used to compile a source code for **different kinds platform**.
- A Cross compiler is a compiler capable of creating executable code for a platform other than one on which compiler is running. E.g. if a compiler runs on a Linux machine and produces executable code for Windows, then it is a cross compiler.

Types of Compiler

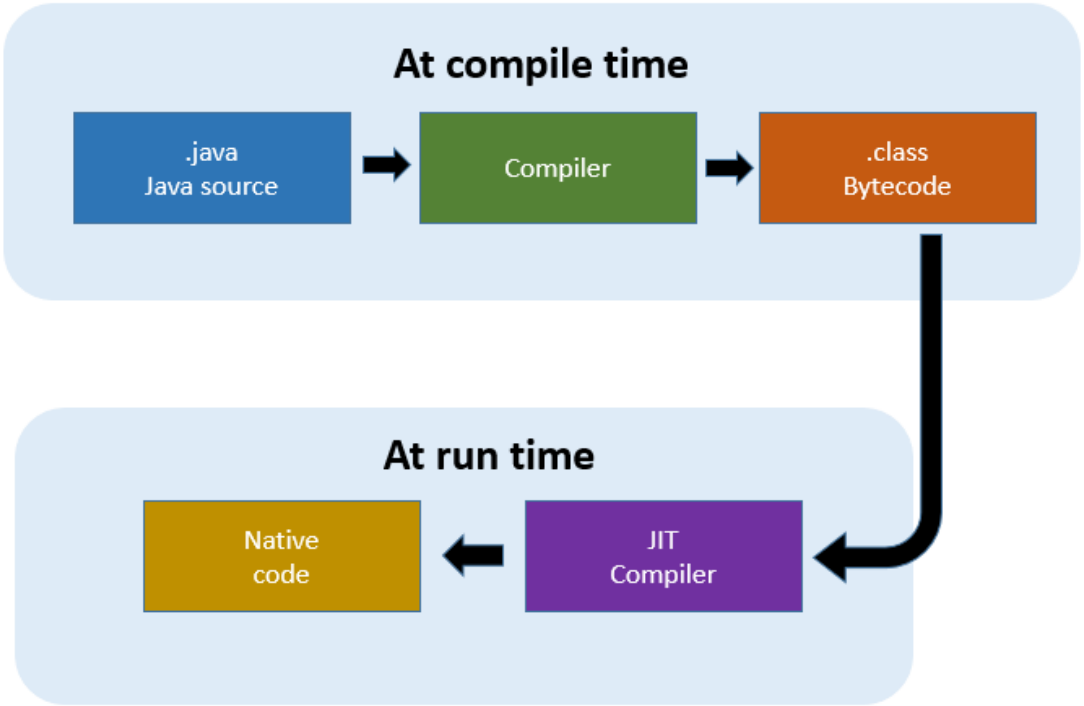
3.Source to Source Compiler :

- A source-to-source translator, source-to-source compiler (**S2S** compiler), **transcompiler**, or **transpiler** is a type of translator that takes the source code of a program written in a programming language as its input and produces an equivalent source code in the same or a different programming language.
- A source-to-source translator converts between programming languages that operate at approximately the same level of abstraction.
- For example, a source-to-source translator may perform a translation of a program from **Python** to **JavaScript.**

Types of Compiler

4. Just in Time Compiler :

- The compiler which converts a code into machine code during execution of program(run-time) rather than before execution.
- It converts .class file(Byte Code) into native code (machine-readable code).



Types of Compiler

5.Incremental Compiler :

- The compiler which compiles only the changed lines from the source code and update the object code. For example, C/C++ GNU compiler.

6.Parallelizing Compiler :

- A Compiler which is specially designed to run in parallel computer architecture is known as parallelizing compiler.

Compiler Construction Tools

Compiler Construction Tools :

1) Scanner Generators
2) Parser Generators
3) Syntax – directed translation engines
4) Automatic Code Generators
5) Data Flow Engine

Compiler Construction Tools

Scanner generators: This tool takes regular expressions as input. For example LEX for Unix Operating System.

Parser generators: A parser generator takes a grammar as input and automatically generates source code which can parse streams of characters with the help of a grammar

Syntax-directed translation engines: These software tools offer an intermediate code by using the parse tree. It has a goal of associating one or more translations with each node of the parse tree.

Automatic code generators: Takes intermediate code and converts them into Machine Language

Data-flow engines: This tool is helpful for code optimization. Here, information is supplied by user and intermediate code is compared to analyze any relation. It is also known as data-flow analysis. It helps you to find out how values are transmitted from one part of the program to another part.

Qualities of Compiler

- Compiler itself must be **bug free.**
- It must generate **correct machine code.**
- Generated machine code **must run fast** (Execution speed).
- **Compilation time** must be **less.**
- Compilation must be **portable.**
- It must be print **good diagnostics** and **Error message.**
- Generated code must work well with **existing debugger.**
- It must have **consistent** and **Predictable optimization.**

References

1. Aho, Lam, Sethi, and Ullman, Compilers: Principles, Techniques and Tools,SecondEdition, Pearson, 2014
2. D. M. Dhamdhere: Systerm Programming, Mc Graw Hill Publication
3. Dick Grune, Henri E. Bal, Jacob, Langendoen: Modern Compiler Design, Wiley India Publication

MU Questions

1) Develop Analysis – Synthesis model of Compiler for a = b + c * 20. Write down explanation and output for each phase of compiler.
2) Difference between Compiler and Interpreter.
3) Explain Cousins of compiler.
4) Give the name of Front End and Back End with explanation, and take one example to show the output of the every phase.

Thanks

Marwadi University

Unit no : 1
Introduction to Compiler
(01CE0601)

Prof. Shilpa Singhal