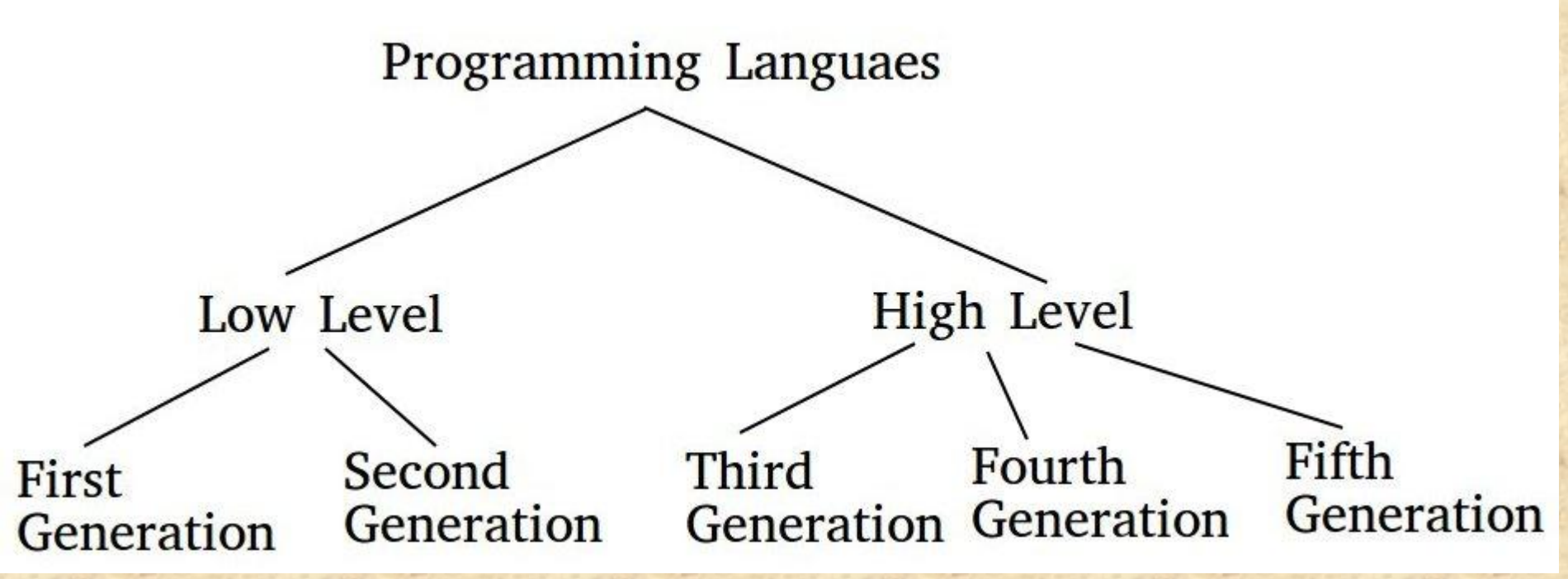
COMPUTER PROGRAMMING

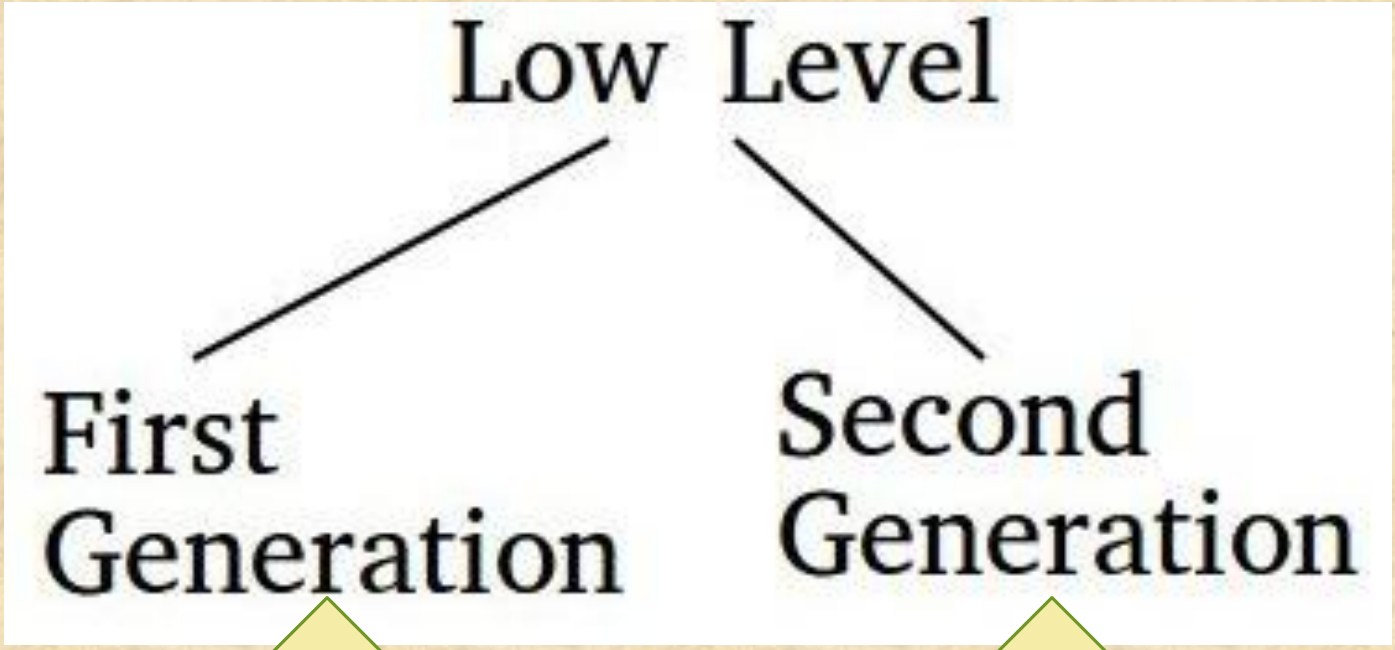
Part 2

PROGRAMMING LANGUAGE GENERATIONS

"…….Programming language is engineered to create a standard form of commands which can be interpreted into a code understood by a machine so that the programmer can control the behavior and output of that machine."



LOW LEVEL LANGUAGE



LOW LEVEL LANGUAGE

Low level computer languages are machine codes or close to it requiring simple straight forward translation. Computer cannot understand or execute instruction given in plain human language. It can only work when instructions are in binary 0s and 1s which is what we call machine language.

- Low level language is machine friendly language.
- It is tough to understand and learn.
- It is complex to debug and maintain comparatively.
- It is machine-dependent.
- Low level language is highly memory efficient.
- It requires simple or no translation.

LOW LEVEL LANGUAGE: THE MACHINE LANGUAGE

Machine Language:

Machine language is basically the only language which the computer can understand. Machine code is represented inside the computer by a string of binary digits consisting of only 0s and 1s.

0000 0101 1100 0001 → String of Binary Digit

A computer can only understand two states:

Presence of Electricity in its Circuit (represented by 1)

Absence of Electricity in its Circuit (represented by 0)

Therefore a digital computer's language or machine language is made of only 0s and 1s.

LOW LEVEL LANGUAGE: THE MACHINE LANGUAGE

Advantages of Machine Language:

- ✓ It is fast as there is no waste of time in code translation.
- ✓ Codes written in machine language can utilize the memory in a very efficient manner.
- ✓ While using machine language, the programmer has the complete control over every aspect of a program's execution as machine language addresses the computer's hardware directly.

LOW LEVEL LANGUAGE: THE MACHINE LANGUAGE

Disadvantages of Machine Language:

- The programmer has to remember all the operation code which are pure binary words.
- Finding errors and bugs in a code which is not working as intended is a complete nightmare.
- Machine language is machine dependent. Instruction set differs from machine to machine. Even any change in any memory controller requires the code to be updated.

LOW LEVEL LANGUAGE: THE ASSEMBLY LANGUAGE

Assembly Language:

To alleviate the problems programmers encounter in machine language programming, the assembly was developed. Though it is also a low level language, it requires a translator program called "Assembler".

The job of assembler was simpler compared to the job done by interpreter or compiler (they translate high level language). It was simpler because in assembly, simple "mnemonics" were introduced to represent their binary counterpart.

These mnemonic codes were short and easy to understand shortened English words.

Example: ADD, SUB, START, LDA, MOV, JMP

LOW LEVEL LANGUAGE: THE ASSEMBLY LANGUAGE

Binary String vs. Mnemonic Code

Machine Language				Assembly Language			
0000	0000	0000	0000	TOTAL	.BLOCK	1	
0000	0000	0000	0010	ABC	.WORD	2	
0000	0000	0000	0011	XYZ	.WORD	3	
0001	1101	0000	0001		LOAD	REGD,	ABC
0001	1110	0000	0010		LOAD	REGE,	XYZ
0101	1111	1101	1110		ADD	REGF,	REGD, REGE
0010	1111	0000	0000		STORE	REGF,	TOTAL
1111	0000	0000	0000		HALT		

LOW LEVEL LANGUAGE: THE ASSEMBLY LANGUAGE

Advantages of Assembly Language:

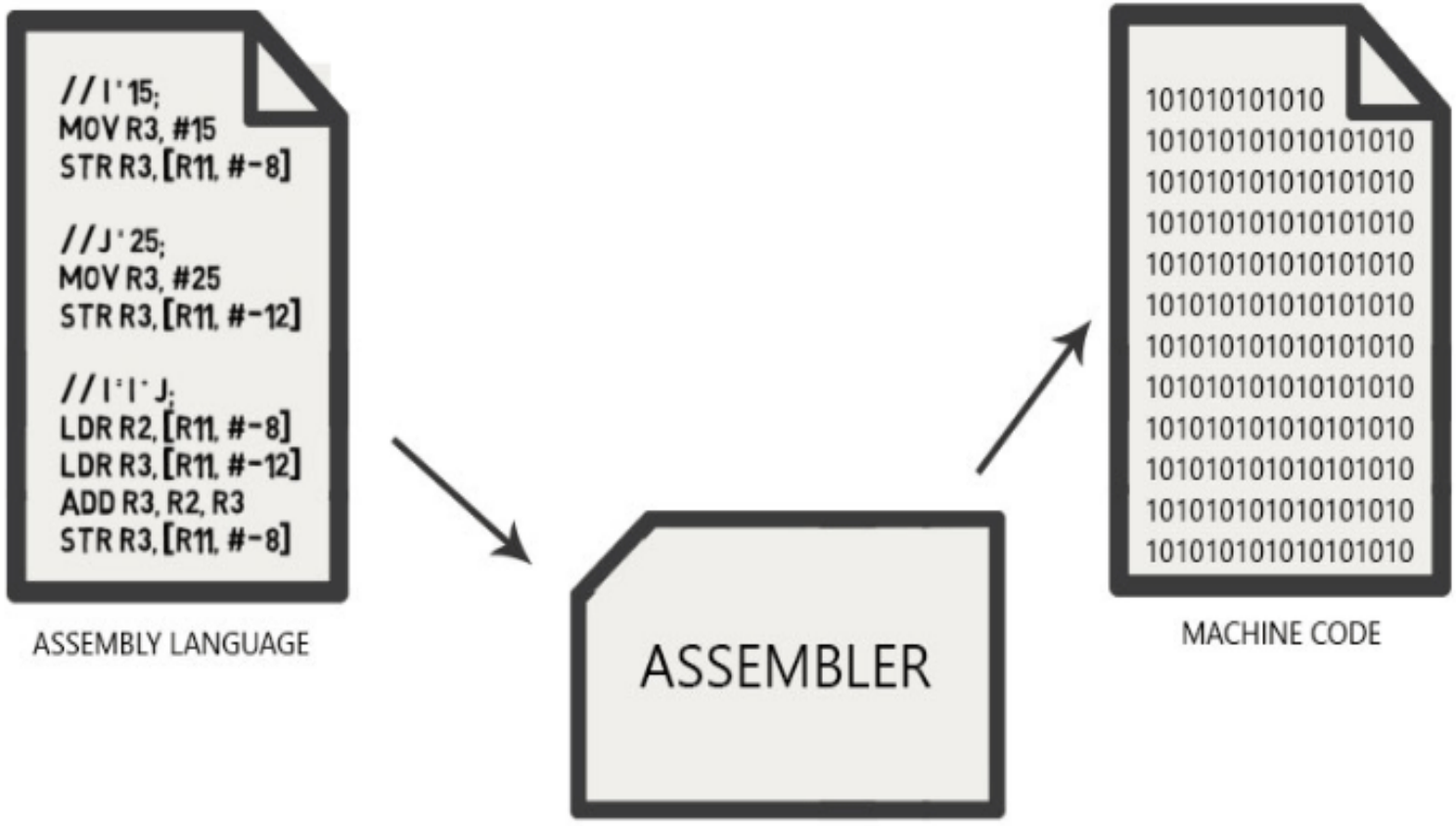
- ✓ Mnemonic codes are easier to memorize and this makes the learning of assembly language an easier task.
- ✓ Finding and correcting errors and bugs is easier compared to machine code.
- ✓ While using assembly language, the programmer has the complete control over hardware and can manage memory efficiently avoiding the more complex machine language.

LOW LEVEL LANGUAGE: THE ASSEMBLY LANGUAGE

Disadvantages of Machine Language:

- Just like machine language, assembly is also machine dependent. Any change in any memory, its controller or any change in main or co-processor may require the code to be changed, too.
- And because of this, the programmer should have detailed knowledge about that particular hardware configuration and the system architecture.
- As it still requires a translator, codes written in assembly take little bit of extra time to get executed.

LOW LEVEL LANGUAGE: THE ASSEMBLY LANGUAGE



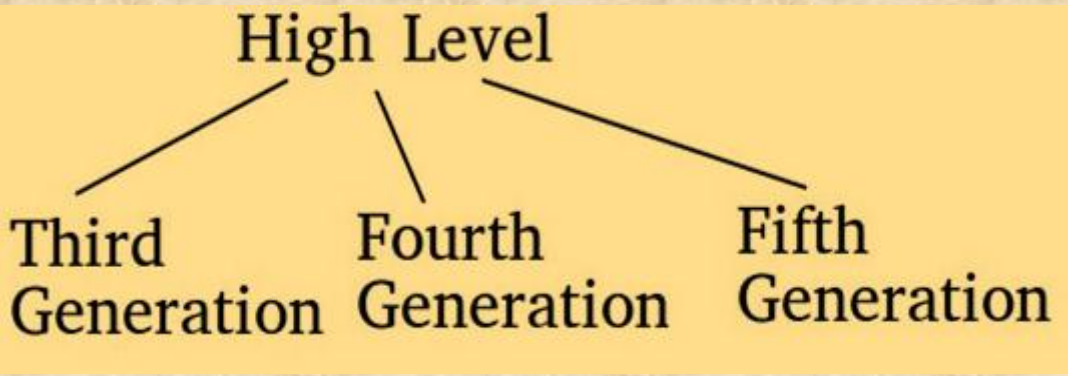
HIGH LEVEL LANGUAGE

High level programming languages bear close resemblance to spoken English language and it is done so programmers can learn and use high level languages with ease. Instead of mnemonic codes, these languages use simple English words and mathematical expressions to write instruction.

Because of its resemblance to human language, high level programming languages requires complex translator programs.

Examples

C, C++, C#, Java, BASIC, Python, Fortran, Pascal, ABAP, ALGOL, OPS5, Mercury, Lisp, Oracle, SQL, COBOL, LANSA, LabVIEW, PLSQL, XBASE++



HIGH LEVEL LANGUAGE

Advantages of High Level Language:

- ✓ Words directly from English dictionary are used as commands and contemporary mathematical symbols are used in expressions in high level language. This is why these languages are easier to learn.
- ✓ Less error prone and finding and debugging errors are easier.
- ✓ Codes are short and fewer and simpler instructions are capable of doing much more work.
- ✓ Machine independent. One code can be translated for many machines depending upon those machines' configurations.
- ✓ As they are programmer friendly, use of high level languages provide better productivity.

HIGH LEVEL LANGUAGE

Disadvantages of High Level Language:

- High level programming languages are often interpreted and It takes additional translation times to translate the source code to machine code.
- They are comparatively slower than low level programs.
- Compared to low level programs, they are generally less memory efficient.
- While using these languages, programmer can’t communicate directly with the hardware as it is done automatically by the language translator itself.

HIGH LEVEL LANGUAGE: THIRD GENERATION

Third generation programming languages or 3GLs are programmer friendly and their focus were more towards machine independence.

Some features of 3GLs are:

- 3GLs supports more data types and various data structures.
- Instead of machine, they were more friendly towards the programmer.
- Non essential details like memory management were taken care of by the computer instead of the programmer.
- Most 3GLs support structured programming.
- Some 3GLs even support object oriented programming.

Examples: ALGOL, BASIC, C, COBOL, Fortran, Java, Pascal, etc.

HIGH LEVEL LANGUAGE: FOURTH GENERATION

Fourth generation programming languages or 4GLs are those which are more advanced compared to other contemporary high level languages. These languages are domain specific and have following features:

- 4GLs focus on large collection of information rather than bit/bytes.
- 4GL include support for database management, report generation, mathematical optimization, GUI development, or web development.
- General purpose 4GLs even support 3GL applications.
- Some 4GLs are simulators and are used for electronic circuit simulation.

Example: Python, Ruby (general purpose), SQL, Ramis (Query) Linc, Oracle Report (report generator), LabVIEW, MultiSim, (simulator)

HIGH LEVEL LANGUAGE: FIFTH GENERATION

Fifth generation languages or 5GLs are a little different than other high level languages. Instead of developing algorithm for the program, programmers make computers to solve a given problem by giving constraints, data and logic.

5GLs are used in the research field of artificial intelligence or AI. Some examples of 5GLs are Mercury, OPS5, ICAD, KL-ONE, etc.

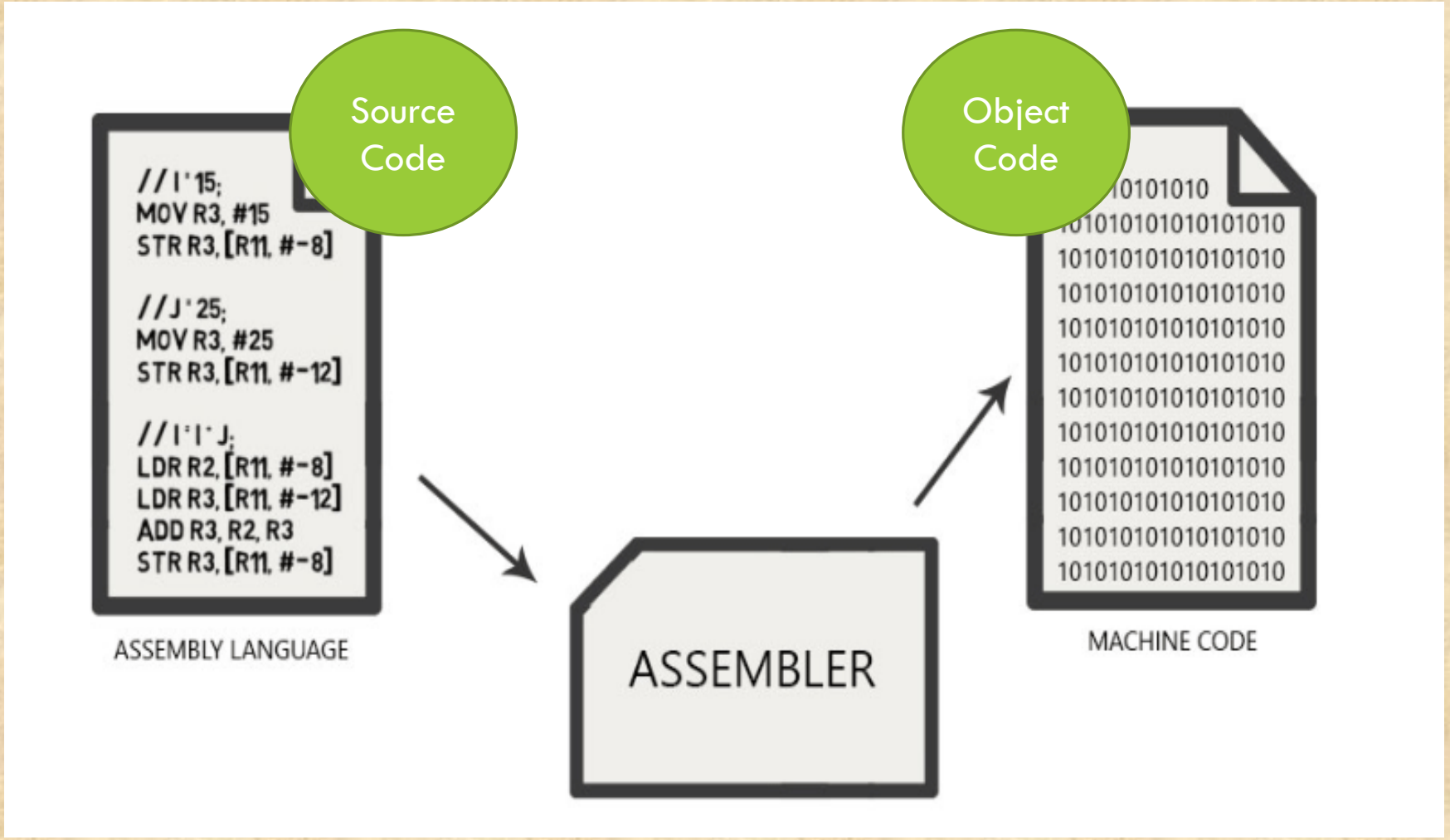
HIGH LEVEL LANGUAGE TRANSLATOR

A source code is a collection of instruction written in programming language and if it is not written in machine language, then the source code has to go through a translation phase so that the machine can have it in machine code/language and can execute the commands in the instructions.

2nd generation assembly language uses a simple translator called assembler whereas all the other high level later generation languages requires complex translators like compiler and interpreter.

- Low Level Language Translator
  - Assembler
- High Level Language Translator
  - Compiler
  - Interpreter

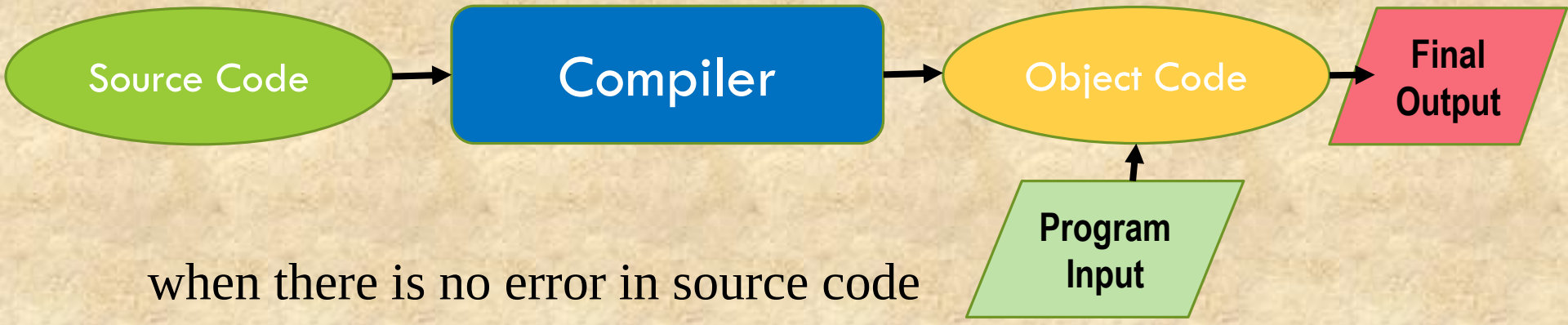
LET'S RECAP A LITTLE



HIGH LEVEL LANGUAGE TRANSLATOR: COMPILER

A compiler is a high level language translator that translates all the instructions in a source code and covert it into object code in machine language before execution.

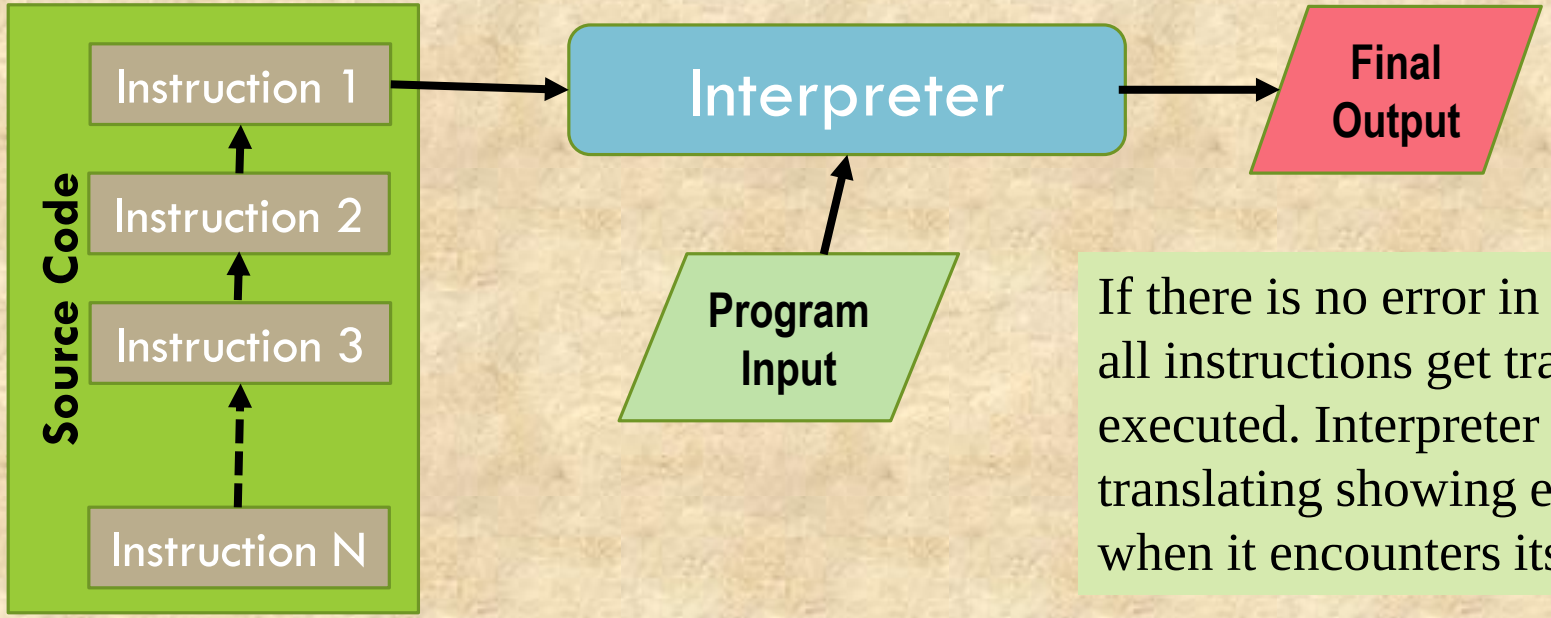
if there is error in source code



when there is no error in source code

HIGH LEVEL LANGUAGE TRANSLATOR: INTERPRETER

An interpreter is also a high level language translator but it differs from compiler as it takes just one instruction from the source code as input, performs translation, executes the command from that translated instruction and then takes next instruction as its input from the source code.



If there is no error in source code, all instructions get translated and executed. Interpreter stops translating showing error message when it encounters its first error.

COMPILER VS. INTERPRETER

Compiler	Interpreter
It takes all instructions from a source code as its input.	It takes instructions one by one, one after another.
All potential errors are shown together before translation.	It shows error upon finding one & doesn't show the next before the correction of the previous one.
For testing & debugging, it is slow.	Good for testing and debugging.
Makes machine code after translation.	No machine code is generated.
As it makes machine code, compiler program is no longer required for subsequent program execution.	Each time the source code requires to be executed, interpreter is required.
It handles logical statements faster.	It gets slower when it has deal with logical statements.

Stay Home, Stay Safe
Always put on a mask when you are in public!