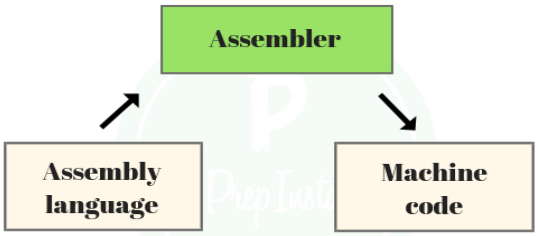
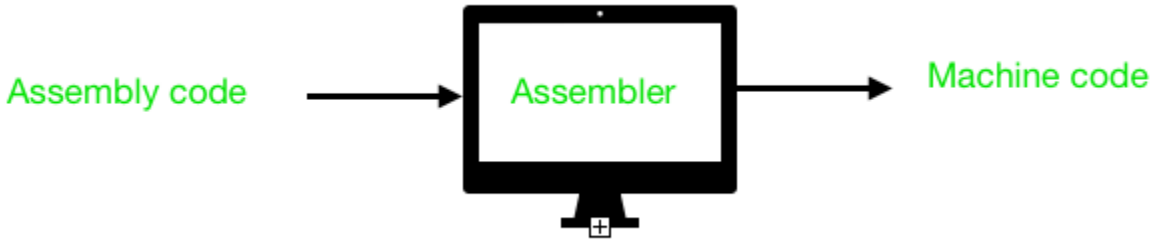
**Assembler**

In computer science, assembler is a program which converts assembly language into machine code. A computer doesn't understand human languages like English or french, but it deals in a much simpler language called binary language, but a programmer can not write the whole program with its complexity in a binary language therefore we need a program that can convert the human written language (assembly language) into binary language, these softwares are called assemblers.

In assembler, a programmer can write a program into sequence of assembler instructions, the sequence of assembler instruction is known as source code and source program.


**Assembler** is a program for converting instructions written in low-level assembly code into relocatable machine code and generating along information for the loader.




It generates instructions by evaluating the mnemonics (symbols) in operation field and find the value of symbol and literals to produce machine code. Now, if assembler do all this work in one scan then it is called single pass assembler, otherwise if it does in multiple scans then called multiple pass assembler. Here assembler divide these tasks in two passes:

- **Pass-1:**
  1. Define symbols and literals and remember them in symbol table and literal table respectively.
  2. Keep track of location counter

3. Process pseudo-operations

- **Pass-2:**
  1. Generate object code by converting symbolic op-code into respective numeric op-code
  2. Generate data for literals and look for values of symbols

Firstly, We will take a small assembly language program to understand the working in their respective passes. Assembly language statement format:

```
[Label] [Opcode] [operand]


Example:  M  ADD  R1, ='3'
where, M - Label; ADD - symbolic opcode;
R1 - symbolic register operand; (='3') - Literal

Assembly Program:
Label  Op-code   operand   LC value(Location counter)
JOHN   START     200
       MOVER     R1, ='3'    200
       MOVEM     R1, X       201
L1     MOVER     R2, ='2'    202
       LTORG                 203
X      DS        1           204
       END                   205
```

Let's take a look on how this program is working:

1. **START:** This instruction starts the execution of program from location 200 and label with START provides name for the program.(JOHN is name for program)
2. **MOVER:** It moves the content of literal(='3′) into register operand R1.
3. **MOVEM:** It moves the content of register into memory operand(X).
4. **MOVER:** It again moves the content of literal(='2′) into register operand R2 and its label is specified as L1.
5. **LTORG:** It assigns address to literals(current LC value).
6. **DS(Data Space):** It assigns a data space of 1 to Symbol X.
7. **END:** It finishes the program execution.

**Working of Pass-1:** Define Symbol and literal table with their addresses.
Note: Literal address is specified by LTORG or END.
**Step-1: START 200** (here no symbol or literal is found so both table would be empty)
**Step-2: MOVER R1, ='3′ 200** ( ='3′ is a literal so literal table is made)

| LITERAL | ADDRESS |
|---|---|
| ='3′ | – – – |

**Step-3: MOVEM R1, X 201**
X is a symbol referred prior to its declaration so it is stored in symbol table with blank address field.

| SYMBOL | ADDRESS |
|---|---|

| | |
|---|---|
| X | – – – |

**Step-4: L1 MOVER R2, =’2′ 202**
L1 is a label and =’2′ is a literal so store them in respective tables

| SYMBOL | ADDRESS |
|---|---|
| X | – – – |
| L1 | 202 |

| LITERAL | ADDRESS |
|---|---|
| =’3′ | – – – |
| =’2′ | – – – |

**Step-5: LTORG 203**
Assign address to first literal specified by LC value, i.e., 203

| LITERAL | ADDRESS |
|---|---|
| =’3′ | 203 |
| =’2′ | – – – |

**Step-6: X DS 1 204**
It is a data declaration statement i.e X is assigned data space of 1. But X is a symbol which was referred earlier in step 3 and defined in step 6.This condition is called Forward Reference Problem where variable is referred prior to its declaration and can be solved by back-patching. So now assembler will assign X the address specified by LC value of current step.

| SYMBOL | ADDRESS |
|---|---|
| X | 204 |
| L1 | 202 |

**Step-7: END 205**
Program finishes execution and remaining literal will get address specified by LC value of END instruction. Here is the complete symbol and literal table made by pass 1 of assembler.

| SYMBOL | ADDRESS |
|---|---|
| X | 204 |
| L1 | 202 |

| LITERAL | ADDRESS |
|---|---|
| =’3′ | 203 |

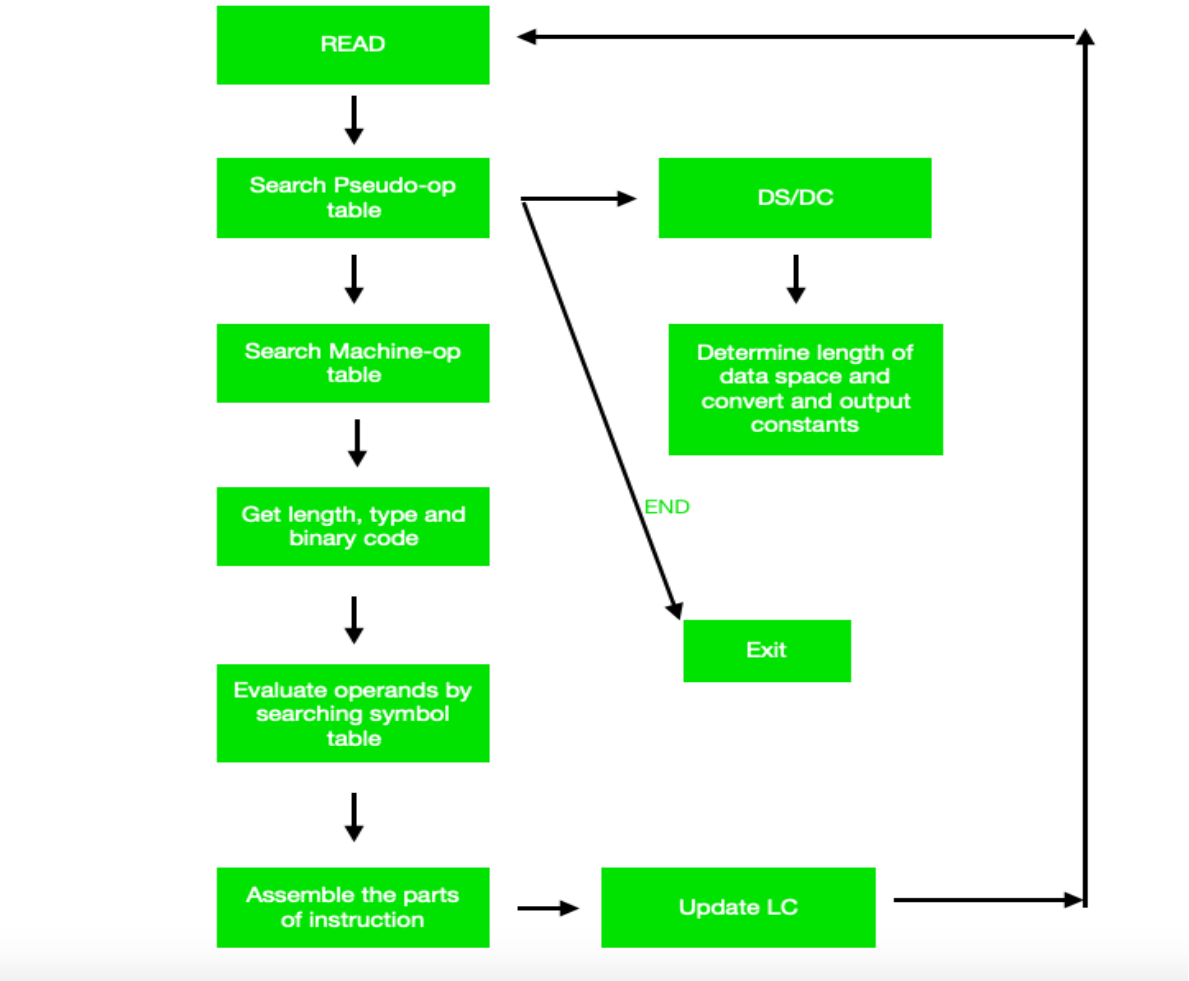
| | |
|---|---|
| ='2′ | 205 |

Now tables generated by pass 1 along with their LC value will go to pass-2 of assembler for further processing of pseudo-opcodes and machine op-codes.

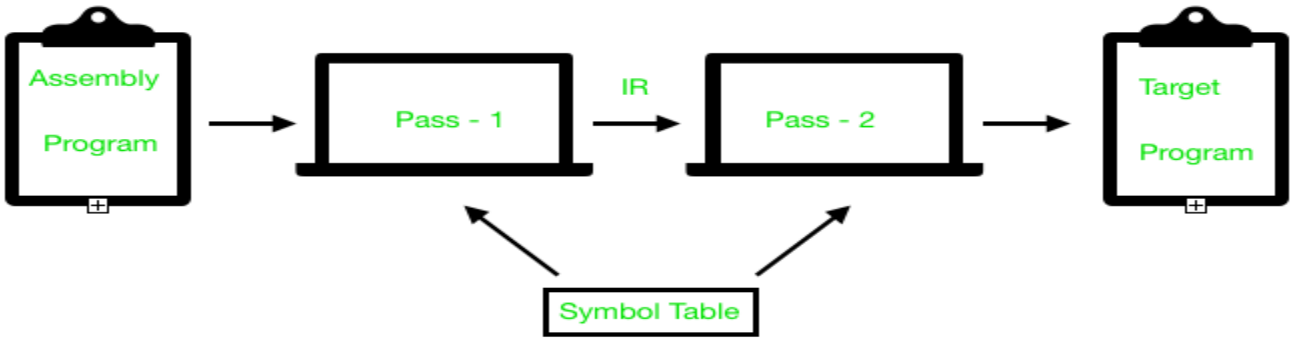
**Working of Pass-2:**
Pass-2 of assembler generates machine code by converting symbolic machine-opcodes into their respective bit configuration(machine understandable form). It stores all machine-opcodes in MOT table (op-code table) with symbolic code, their length and their bit configuration. It will also process pseudo-ops and will store them in POT table(pseudo-op table).
Various Data bases required by pass-2:

```
1. MOT table(machine opcode table)
2. POT table(pseudo opcode table)
3. Base table(storing value of base register)
4. LC ( location counter)
```

Take a look at flowchart to understand:




As a whole assembler works as:



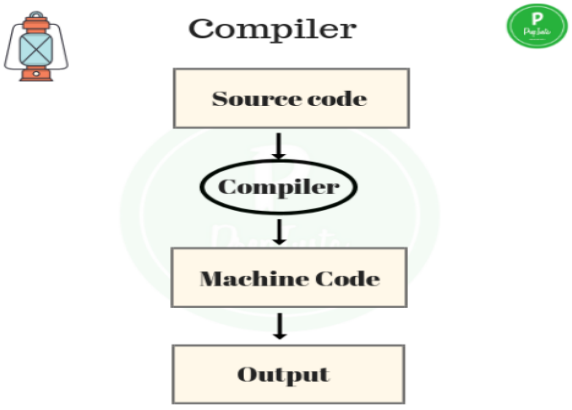
# Assembler Design

- Machine Dependent Assembler Features
  - instruction formats and addressing modes
  - program relocation
- Machine Independent Assembler Features
  - literals
  - symbol-defining statements
  - expressions
  - program blocks
  - control sections and program linking
- Assembler design Options
  - one-pass assemblers
  - multi-pass assemblers

**Compiler**

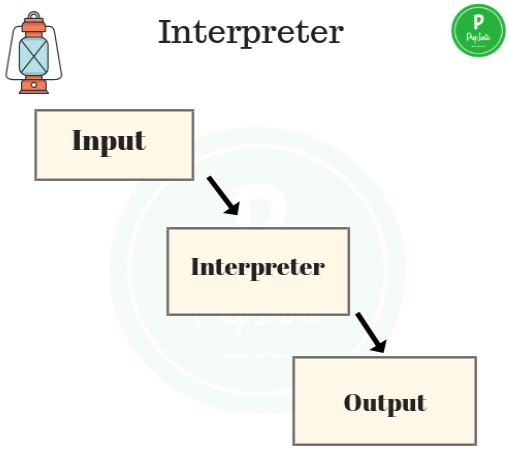
A Compiler is a program that converts a number of statement of program into binary language, but it is more intelligent than interpreter because it goes through the entire code at once and can tell the possible errors and limits and ranges.But this makes it's operating time a little slower.it is platform-dependent.its help to detect error and get displayed after reading the entire code by compiler.

In other words we can say that, "Compilers turns the high level language to binary language or machine code at only time once", it is known as Compiler.




**Interpreter**

An interpreter is also a program like a compiler that converts assembly language into binary but an interpreter goes through one line of code at a time and executes it and then goes on to the next line of the code and then the next and keeps going on until there is an error in the line or the code has completed.It is 5 to 25 times faster than a compiler but it stops at the line where error occurs and then again if the next line has an error too, where as a compiler gives all the errors in the code at once.if changes are made on that code which is already compiled then the changed code will need to be compiled and added to compiled code or the entire code need to be re-compiled. Also, a compiler saves the machine codes for future use permanently but an interpreter doesn't, but an interpreter occupies less memory.




Interpreter is differ from compiler such as,

- Interpreter is faster than compiler.
- It contains less memory.
- Interpreter executes the instructions in to source programming language.

There are several types of interpreter:

- Syntax-directed interpreter
- Threaded interpreter
- Bytecode interpreter

**Linker**

For a code to run we need to include a header file or a file saved from the library which are pre-defined if they are not included in the beginning of the program then after execution the compiler will generate errors, and the code will not work.

Linker is a program that holds one or more object files which is created by compiler, combines them into one executable file.Linking is implemented at both time,load time and compile time. Compile time is when high level language is turns to machine code and load time is when the code is loaded into the memory by loader.

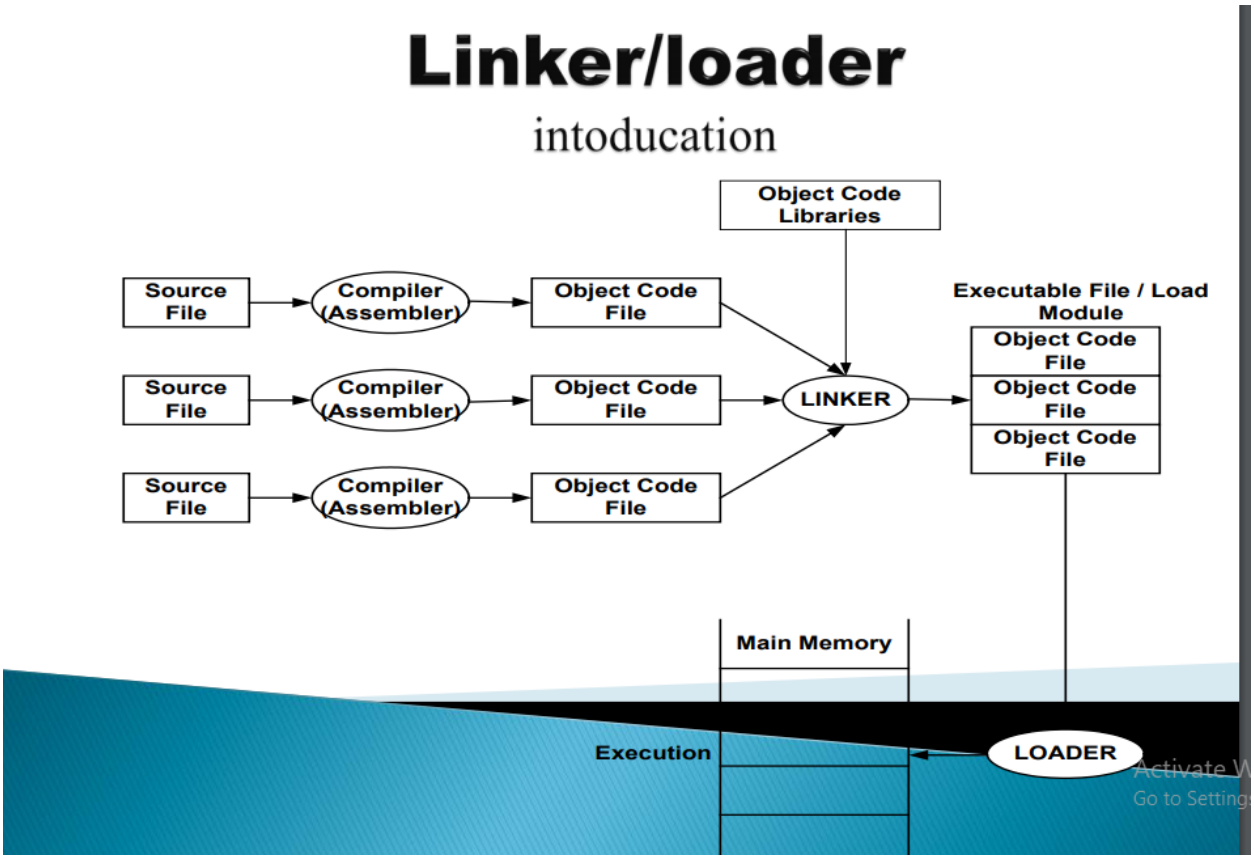
Linker is of two types:

**1.Dynamic Linker:-**

- It is implemented during run time.
- It requires less memory.
- In dynamic linking there are many chances of error and failure chances.
- Linking stored the program in virtual memory to save RAM,So we have need to shared library

**2.Static Linker:-**

- It is implemented during compilation of source program.
- It requires more memory.
- Linking is implemented before execution in static linking.
- It is faster and portable.
- In static linking there are less chances to error and No chances to failure.

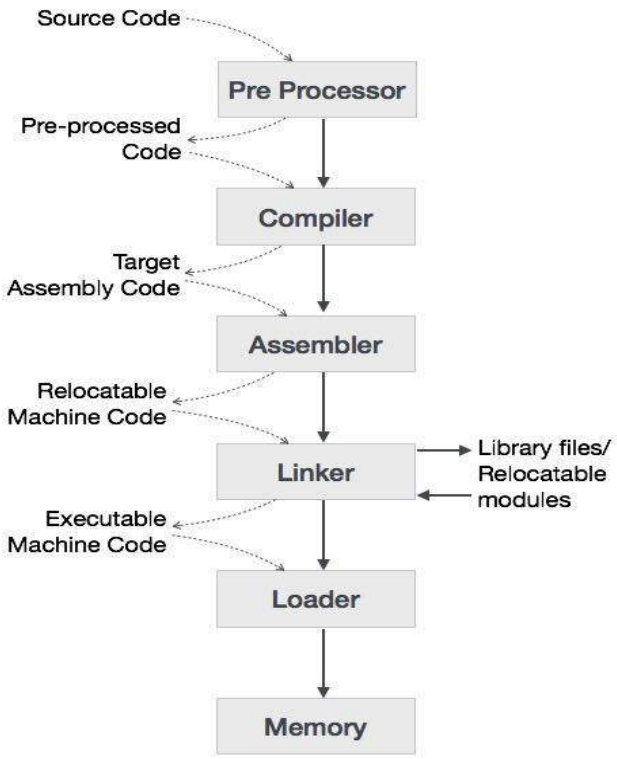



**Loader**

Loader is a program that loads machine codes of a program into the system memory.It is part of the OS of the computer that is responsible for loading the program. It is the bare beginning of the execution of a program. Loading a program involves reading the contents of executable file into memory. Only after the program is loaded the operating system starts the program by passing control to the loaded program code. All the OS that support loading have loader and many have loaders permanently in their memory.

**Language Processing System**

We have learnt that any computer system is made of hardware and software. The hardware understands a language, which humans cannot understand. So we write programs in high-level language, which is easier for us to understand and remember. These programs are then fed into a series of tools and OS components to get the desired code that can be used by the machine. This is known as Language Processing System.




The high-level language is converted into binary language in various phases. A **compiler** is a program that converts high-level language to assembly language. Similarly, an **assembler** is a program that converts the assembly language to machine-level language.

Let us first understand how a program, using C compiler, is executed on a host machine.

- User writes a program in C language (high-level language).
- The C compiler, compiles the program and translates it to assembly program (low-level language).
- An assembler then translates the assembly program into machine code (object).
- A linker tool is used to link all the parts of the program together for execution (executable machine code).
- A loader loads all of them into memory and then the program is executed.

Before diving straight into the concepts of compilers, we should understand a few other tools that work closely with compilers.

**Preprocessor**

A preprocessor, generally considered as a part of compiler, is a tool that produces input for compilers. It deals with macro-processing, augmentation, file inclusion, language extension, etc.