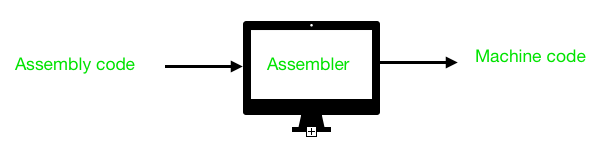
**Assembler** is a program for converting instructions written in low-level assembly code into relocatable machine code and generating along information for the loader.



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:

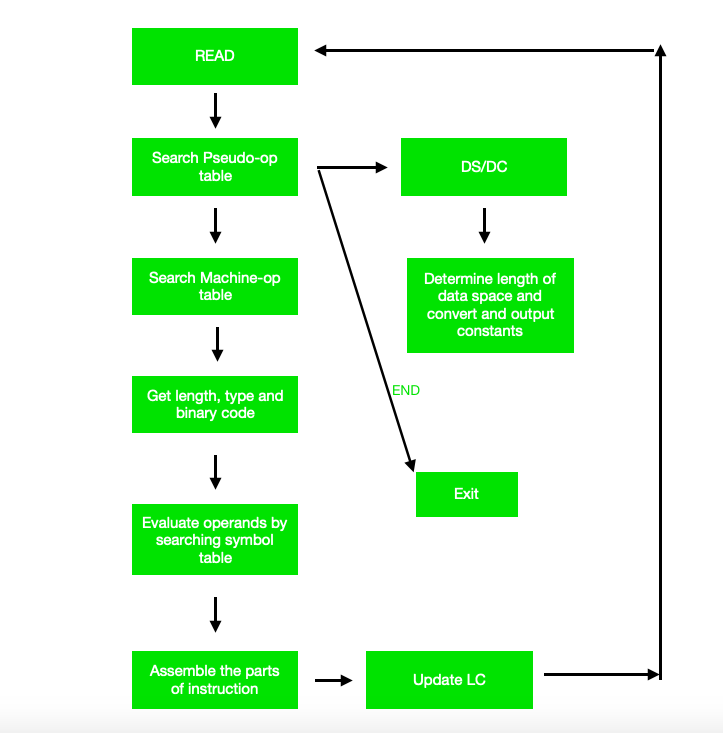
1. MOT table(machine opcode table)

2. POT table(pseudo opcode table)

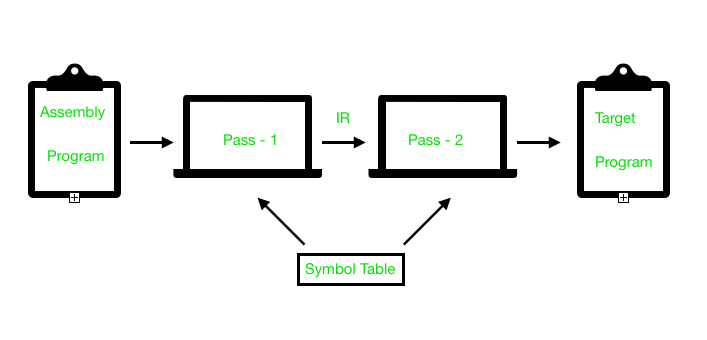
3. Base table(storing value of base register)

4. LC ( location counter)

Take a look at flowchart to understand:



As a whole assembler works as:



Online Notes

<http://www.csie.ntnu.edu.tw/~ghhwang/course_slices/system_software/Chapter2.pdf>

<http://osnet.cs.nchu.edu.tw/powpoint/SP93_2/Chapter%202.pdf>

<http://www.wbuthelp.com/chapter_file/2677.pdf>

<https://www.unf.edu/~cwinton/html/cop3601/s10/class.notes/asm1-Pass1&2.pdf>

Video Links

<https://www.youtube.com/watch?v=GWIcu5edovU>

<https://www.youtube.com/watch?v=a8hXmcdsdTQ>

<https://www.youtube.com/watch?v=HD1gmULuYks>