**Assembler and assembly language**

**Assembler**

An assembler is a program that translates the assembly language to the machine language, there are several different assemblers that depend on the target system’s **ISA**:

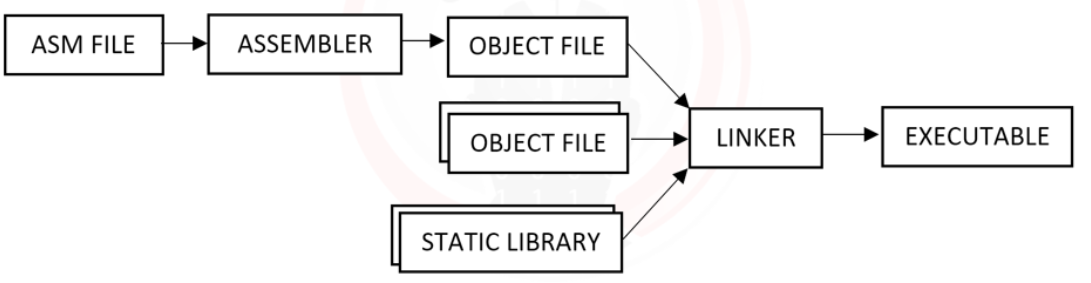
* Microsoft macro assembler (MASM): x86 assembler that uses the intel syntax for MS-DOS and Microsoft windows.
* GNU assembler (GAS): used by the GNU project, default back-end of GCC.
* Netwide assembler (NASM): x86 architecture used to write 16-bit and 32-bit (IA-32) and 64-bit (x86-64) programs, one of the most popular assemblers for Linux.
* Flat assembler (FASM): x86 supports intel-style assembly language on the IA-32 and x86-64.

When source code file is assembled, the resulting file is called **Object file**. It is a binary representation of the program.

Once the assembler has created the object file, a **linker** is needed in order to create the actual executable file. What a linker does is take one or more object files and combine them to create the executable file.

An example of these object files is the kernel32.dll and user32.dll which are required to create windows executable that accesses certain library

The process from the assembly code to the executable file can be represented as:



**NASM**

The assembler we are going to use is **NASM**, and to make things easier, we are going to use NASM-X project. It is a collection of macros, include and examples to help NASM programmers develop applications.

NASM installation instructions:

1. NASM-X Download link: <https://sourceforge.net/projects/nasmx/>
2. Extract NASM-x files and save it to a folder ex: C:\nasmx
3. Modify windows environment variables and add NASM-X binaries, C:\nasmx\bin
4. Try the configuration and open CMD then run
   1. setpath.bat [Enter].

[you will see something like this, in case of everything is running well]

1. Navigate to the C:\nasmx\demos and edit windemos.inc
   1. Comment the following line: %include ‘’.
   2. Adding this right after: “C:\nasmx\inc\nasmx.inc”
2. Verify that everything is configured:
   1. Open terminal and navigate to the folder: C:\nasmx\demos\win32\DEMO1

Generate executable file from assemble file:

1. Assemble demo1.asm:
   1. Command: nasm -f win32 demo1.asm -o demo1.obj [Enter].
2. Executable file with linker:
   1. Command: GoLinker.exe /entry \_main demo1.obj kernel32.dll user32.dll

Note: if the command succeeded, you should see a new file names demo1.exe

**ASM Basics**

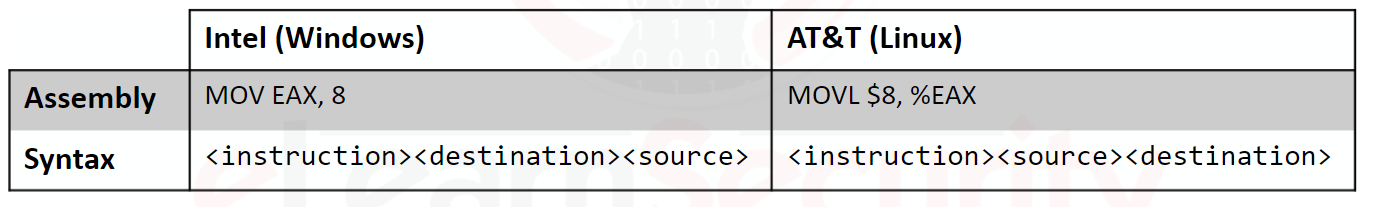
Most instructions have two operands and fall into one of the following classes:

1. Data transfer:
   1. MOV
   2. PUSH
   3. POP
   4. XCHG
2. Arithmetic:
   1. ADD
   2. SUB
   3. MUL
   4. XOR
   5. NOT
3. Control Flow:
   1. CALL
   2. RET
   3. LOOP
   4. Jcc
      1. Where cc is any condition
   5. Other:
      1. STI
      2. CLI
      3. IN
      4. OUT

Example on simple assembly code

mov EAX,2 ; store 2 in EAX  
mov EBX,5 ; store 5 in EBX  
add EAX,EBX ; do EAX = EAX + EBX  
 ; now EAX contains the results

Intel vs AT&T



AT&T

AT&T puts a percent sign (%) before registers name and a dollar sign ($) before numbers, and defines the operand size: Q (quad - 64bits), L (long – 32 bits), W (word – 16 bits), B (Byte – 8 bits)

**CALL**

Subroutines are implemented by using the **CALL** and **RET** instruction pair.

The **CALL** instruction pushes the current instruction pointer (**EIP**) to the stack and jumps to the function address specified, Whenever the function executes the **RET** instruction, the last element is popped from the stack, and the CPU jumps to the address.

[Example on **CALL**]

