Assembly language

Tuesday, January 17, 2017 8:20 PM

A processor can understand and execute machine languages. Instructions are simple binary numbers stored in the computer.

We make use of symbolic name or mnemonic of each instruction to make a symbolic program. When we incorporate this with symbolic addresses, this allows us to have *assembly language.*

Programs written in assembly are translated into machine language by an **assembler.**
NOT A COMPILER.
Compiler and assembler differ in that each input statement does NOT correspond to a single machine instruction or fixed sequence of instructions

Assembler is a software utility that takes an assembly program as input and produces object code as output.

- Object code is a binary file (machine language representation of source code)
- Can be done generally with one-pass or two-pass
    - Two-pass is most common
        - On first pass, the assembler is concerned with label definitions (construct symbol table)
        - On second pass, each instruction is translated to appropriate binary code
- Turned into executable code by the linker

Assembly language and machine language are **NOT** the same.

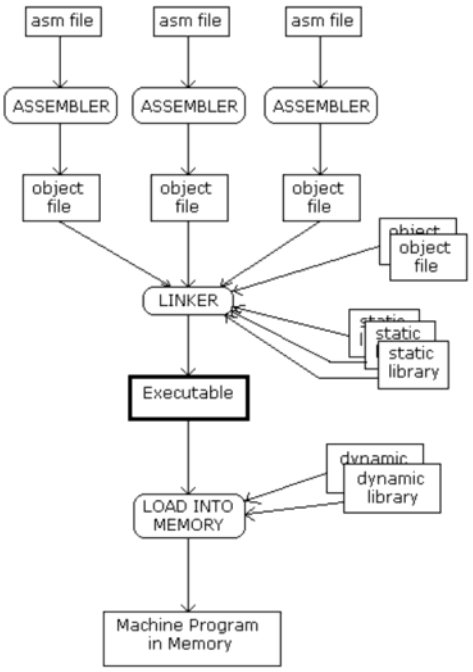
- Machine language consists of instructions directly executable by the processor.
    - Contains opcode, operand references, and other bits for execution (like flags)
- Assembly makes greater use of symbolic names, assigning names to memory locations, and specific instruction location
    - Includes statements that are not directly executable but serve as instructions to the assembler.
    - These statements are called directives, or pseudo-instructions
        - Define constants
        - Initialize areas of memory
        - Place tables or fixed data in memory
        - Allow references to other programs
    - Macro definitions: similar to subroutines; when assembler encounters macro, it replaces macro call with macro itself
        - Handled by the assembler at assembly time (as opposed to subroutines handling by hardware at runtime)

The topic of x86 assembly language programming is messy because:

- **There are many different assemblers out there**: MASM, NASM, gas, as86, TASM, a86, Terse, etc. All use radically different assembly languages.
- **There are differences in the way you have to code for Linux, OS/X, Windows**, etc.
- **Many different object file formats exist**: ELF, COFF, Win32, OMF, a.out for Linux, a.out for FreeBSD, rdf, IEEE-695, as86, etc.
- **have to know some technical details about how libraries are linked**, and not all linkers work the same way.

Types of assemblers:

- **MASM**, the Microsoft Assembler. It outputs OMF files (but Microsoft's linker can convert them to win32 format). It supports a massive and clunky assembly language. **Memory addressing is not intuitive.** The directives required to set up a program make programming unpleasant.
- **GAS**, the GNU assembler. This uses the rather ugly AT&T-style syntax so many people do not like it; however, you can configure it to use and understand the Intel-style. It was designed to be part of the back end of the GNU compiler collection (gcc).
- **NASM**, the "Netwide Assembler." It is free, small, and best of all it can output zillions of different types of object files. The language is much more sensible than MASM in many respects



Linker: combines one or more files containing object code from separately compiled program modules into a single file

Loader: program routine that copies an executable program into memory for execution

Assembly was the precursor to high-level languages. Mostly used for system programs such as compilers and I/O routines

Assembly language statement has 4 elements:

- Label: defines label as equivalent to the address into which the first byte of the object code generated for that instruction will be loaded
  - Frequently used in branch instructions
- Mnemonic: name of the operation or function of the assembly language statement
  - Can be machine instruction, assembler directive, or a macro
- Operand: identifies an immediate value, register value, or memory location
  - Assembly language statement may refer to register operand by name
- Comment: code ignored by assembler
  - Signified by semicolon (;) for most x86