EE 5201 - Computer Architecture

Assembly Programming

H. Asela
Department of Electrical and Information Engineering
University of Ruhuna

Learning Objectives

Demonstrate the ability to write simple assembly programs using instruction set of Intel x86 via assembly programs (using NASM for Linux) using ISA aspects such as I/O devices, Stack, Interrupts, Procedures, arithmetic.

Evaluation

- Take Home Programming Assignments
- Quizzes
- Final Exam

References

Sivarama P. Dandamudi (2005): Guide to Assembly Language Programming in Linux. Springer-Science Inc. NJ-USA

Introduction to Assembly Programming

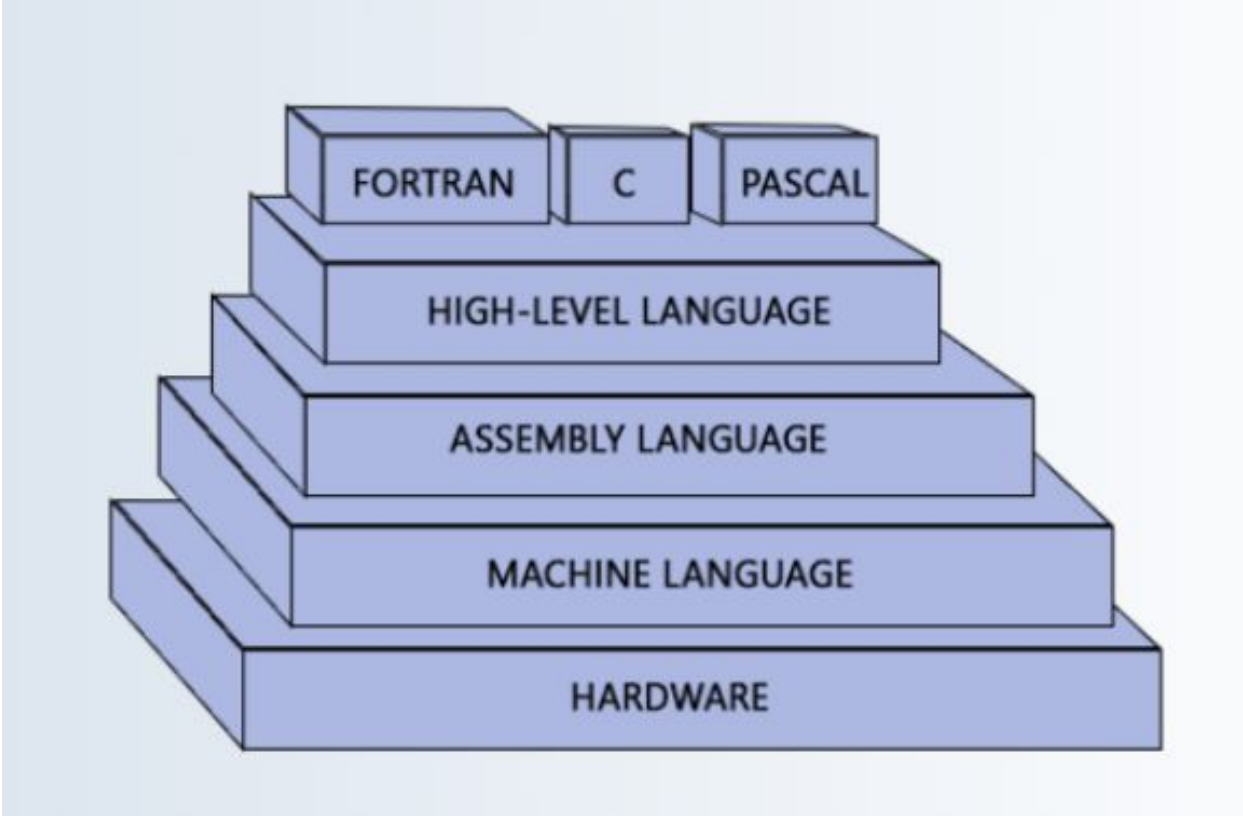
An assembly language is a low-level programming language designed for a specific type of processor.

Assembly language is used primarily for direct hardware manipulation, access to specialized processor instructions, or to address critical performance issues.

Typical uses are device drivers, low-level embedded systems, and real-time systems.

A processor understands only machine language instructions, which are 1’s and 0’s

However, machine language is too obscure and complex to understand for software development, so low-level assembly language is used.


FORTRAN
C
PASCAL
HIGH-LEVEL LANGUAGE
ASSEMBLY LANGUAGE
MACHINE LANGUAGE
HARDWARE

Machine Language

Machine language is a collection of binary digits or bits that the computer reads and interprets. It is directly executed by hardware.

Machine language is the only language a computer is capable of understanding.

A computer cannot directly understand the programming languages used to create computer programs, so the program code must be compiled.

Once a program's code is compiled, the computer can understand it because the program's code is turned into machine language.

Assembly Language

Assembly language has syntaxes similar to English, but more difficult than high-level programming languages.

Assembly code is **NOT** portable across architectures . (Different ISAs, different assembly languages)

To program in assembly language, one should have understood at hardware level like computer architecture, registers, etc.

There is one-to-one correspondence with machine language instructions.

High Level Languages Vs Assembly Language

High Level Language	Assembly Language
It need an compiler/interpreter for conversion	It need an assembler for conversion
machine independent	Machine dependent
English statements are used	Mnemonics codes are used
Difficult to access hardware components	Easy to access hardware component
More compact code	No compactness

High-level program

```
class Triangle {
  ...
  float surface()
    return b*h/2;
  }
```

Low-level program

```
LOAD r1,b
LOAD r2,h
MUL r1,r2
DIV r1,#2
RET
```

Executable Machine code

```
0001001001000101
0010010011101100
10101101001...
```

Why Learn Assembly Language

- Complete control over a system’s resources
- Understanding processor and memory functionality
- Accessibility to system hardware
- Space and time efficiency
- Assembly language is transparent

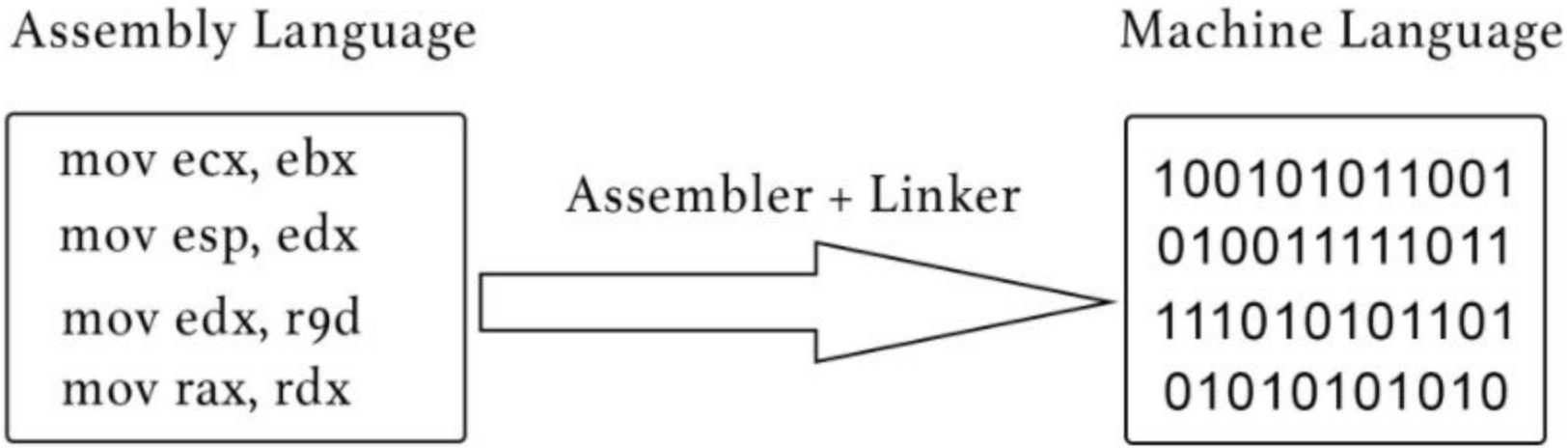
Assembler

An assembler is a program that converts assembly language into machine code.

It takes the basic commands and operations from assembly code and converts them into binary code that can be recognized by a specific type of processor.

Assembly code consists of lots of ASCII characters; this would be stored in a file and called "source code".

This then forms the input to a assembler which translates the "source code" into machine code. or object code.


Assembly Language
mov ecx, ebx
mov esp, edx
mov edx, r9d
mov rax, rdx
Assembler + Linker
Machine Language
100101011001
010011111011
111010101101
01010101010

Popular Assemblers

- MASM (Microsoft Macro Assembler)
- NASM (Netwide Assembler)
- TASM (Turbo Assembler)
- GNU Assembler

Assembling and linking assembly language programs

Here we will be using NASM

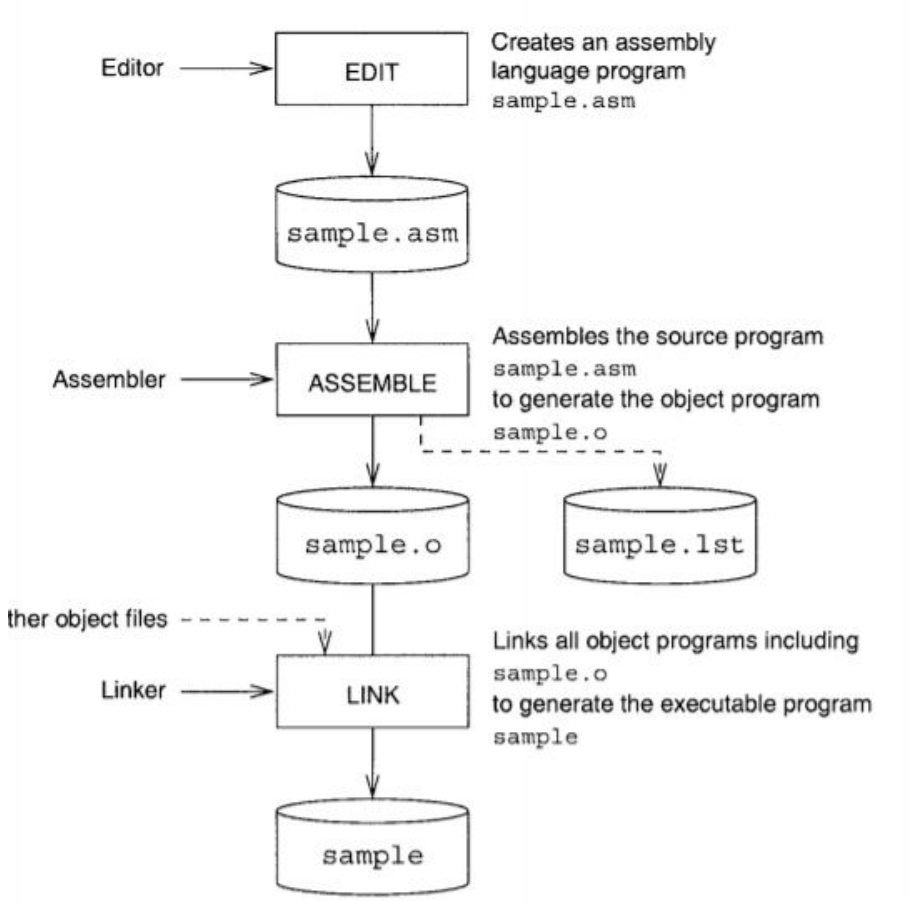
The general format to assemble a program is,

nasm -f <format> [-o <object-file>] [-l <list-file>] <source-file>

After successfully assembling the source program, NASM generates an object file with the same file name as the source file but with . o extension.

To link the object file,

ld [-o <executable-file>] <object-file>



Reference : Guide to Assembly Language Programming in Linux. Springer-Science Inc. NJ-USA

Thank you!