Linux ASM Introdcution

This document is an introduction to Linux ASM. The reason why this exists is because when I was in my Assembler class in college, I found there was no real central location for x86-64 examples. This is an attempt to collate information I have accquired to make other people's lives easier. Code provided is by no means the best version out there. My approach is to learn how to first stand, then walk and then to run. To be clear, x86 describes the umbrella of architectures that was originally created in the early 1970's by Intel. More specifically in the scope of this series, we will be focusing on the x86-64 branch, which only deals with 64-bit instructions.

First off, what is assembler? It is the mnemonic programming language that layers machine code and is below higher level languages such as C/C++. Later on, we will learn how to interface assembler with C code and vice versa. Assembler (also known as assembly language) was created because it is easier for programmers to remember short pseudo instructions rather than hexidecimal operation codes.

For example, these two code segments are identical. Which one would you rather write? As a preface, this and further documents will be using Intel sytle as it much easier to read and write in my opinion. This does not mean that writing in the standard AT&T syntax is "unacceptable", it just means the code provided will not directly translate into your program.

```
; Mnemonic to move the content of r9 into the r8 register
mov r8, r9
```

```
; Opcode (operation code)
; 88 /r
```

Writing in pure opcodes is error prone and is highly discouraged unless there is some inherent advantage, most notably an increase in computation speed.

External Links

x86 Instruction Set Reference