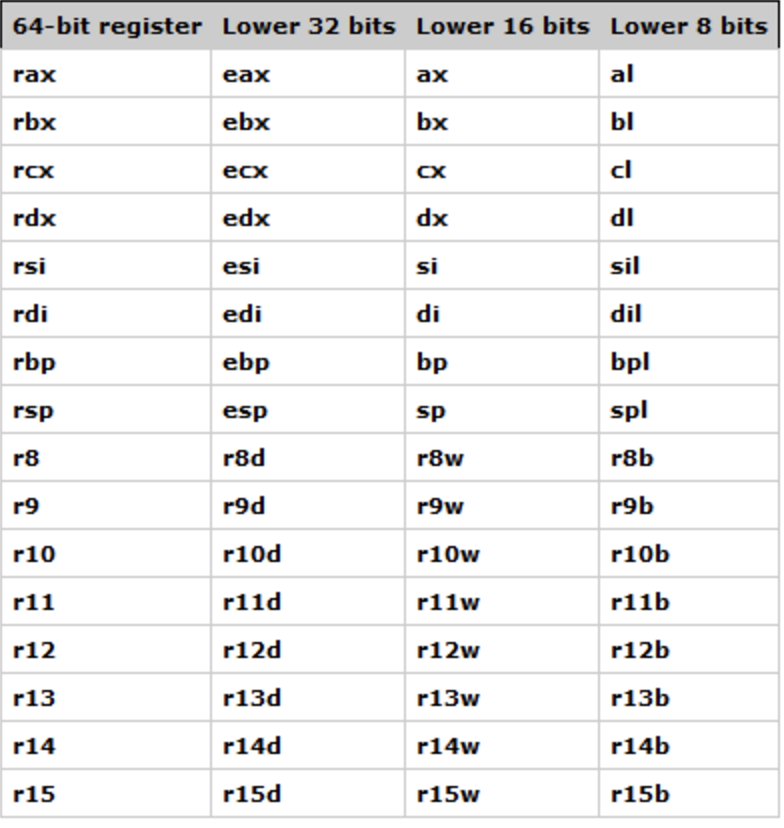
Getting Started Demo/Tutorial

First assembler language program (NASM/Linux Platform)

References

* <http://www.jegerlehner.ch/intel/IntelCodeTable.pdf> (I32 instruction set)
* <http://pacman128.github.io/static/pcasm-book.pdf> (general reference book)
* <http://blog.rchapman.org/posts/Linux_System_Call_Table_for_x86_64/> (64 bit linux sys call)
* <http://syscalls.kernelgrok.com> (32 bit linux sys call)



### Install and check NASM

|  |  |
| --- | --- |
| Linux terminal window | Install and check location:   * sudo apt-get install nasm * which nasm |
| Notes from <http://www.nasm.us/doc/nasmdoc0.html> (Netwide Aseembler document)  To assemble a file, you issue a command of the form   * nasm −f [−o ]   For example,   * nasm −f elf64 myfile.asm   will assemble myfile.asm into an ELF 64bit object file myfile.o.  To produce a listing file, with the hex codes output from NASM displayed on the left of the original sources, use the −l option to give a listing file name, for example:   * nasm −f elf64 myfile.asm −l myfile.lst   To get further usage instructions from NASM, try typing   * nasm –hf   will also list the available output file formats, and what they are.  If you use Linux but aren’t sure whether your system is a.out or ELF, type   * file /usr/bin/nasm   (or in the directory in which you put the NASM binary when you installed it). | |

For Mac OSx (install homebrew first): https://raw.githubusercontent.com/Homebrew/install/master/install)"

* /usr/bin/ruby -e "$(curl -fsSL https://raw.githubusercontent.com/Homebrew/install/master/install)"
* brew install nasm
* nasm –v
* nasm –hf

### Create a 32-bit program (with system call)



ex1.asm

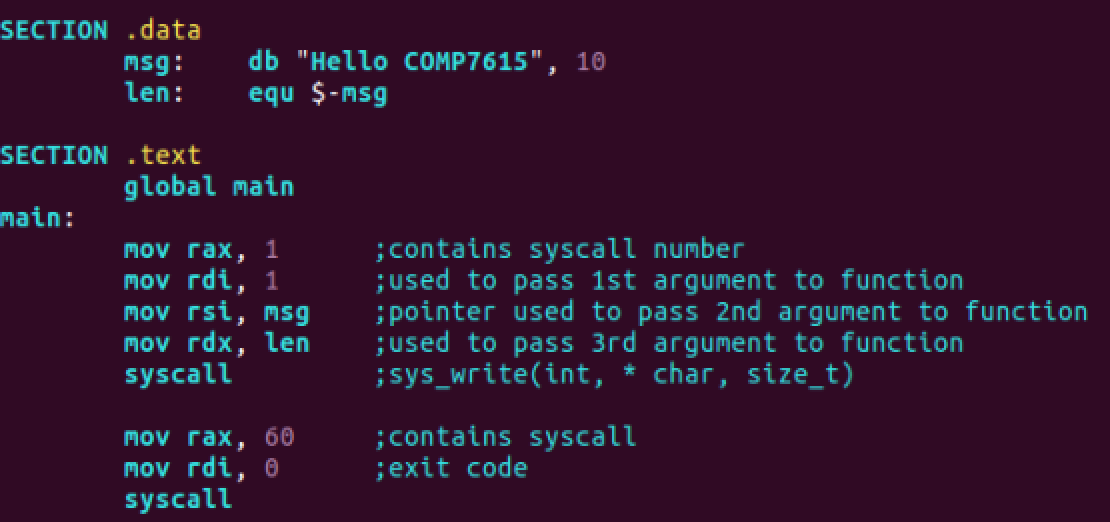
To compile, link and run, a 32-bit program for 64-bit machine:

* nasm –f elf64 ex1.asm (-l ex1.lst)
* gcc –o ex1 ex1.o

OR for gcc 5.0 and up:

* gcc -fno-pie -no-pie ex1.o
* ./a.out

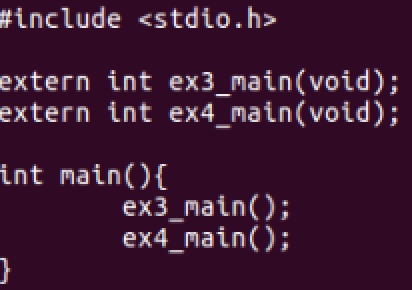
### Create a 64-bit program (with system call)



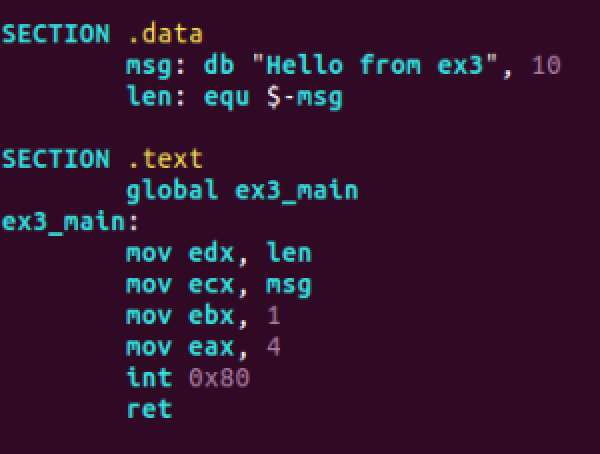
To compile, link and run, a 64-bit program for 64-bit machine:

* nasm –f elf64 ex2.asm (-l ex2.lst)
* gcc ex2.o
* ./a.out

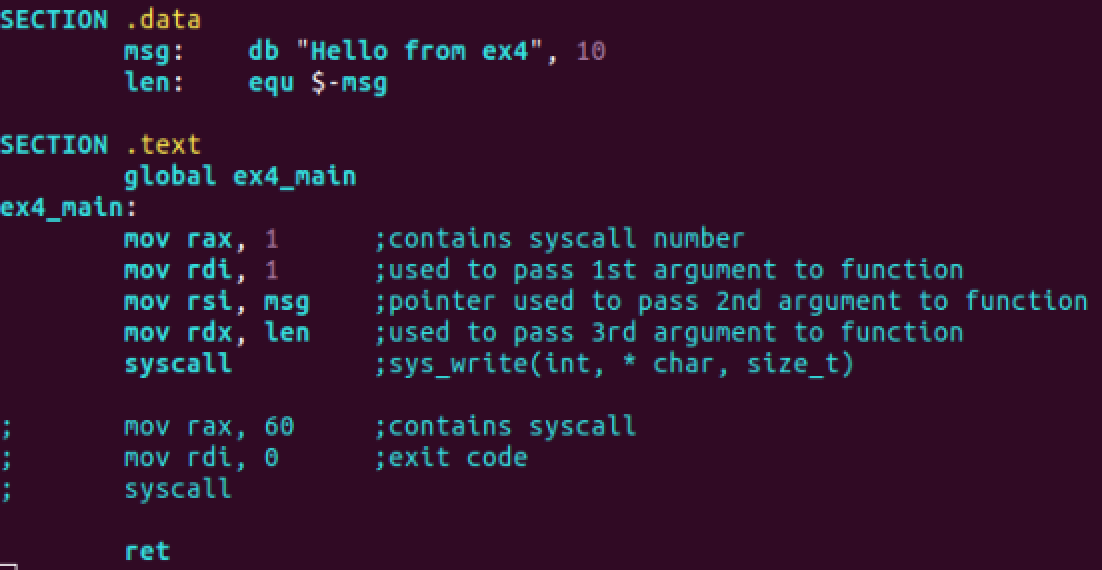
### Create functions (both 32-bit or 64-bit ones) that can be called from C driver.



driver.c



ex3.asm (32-bit function)

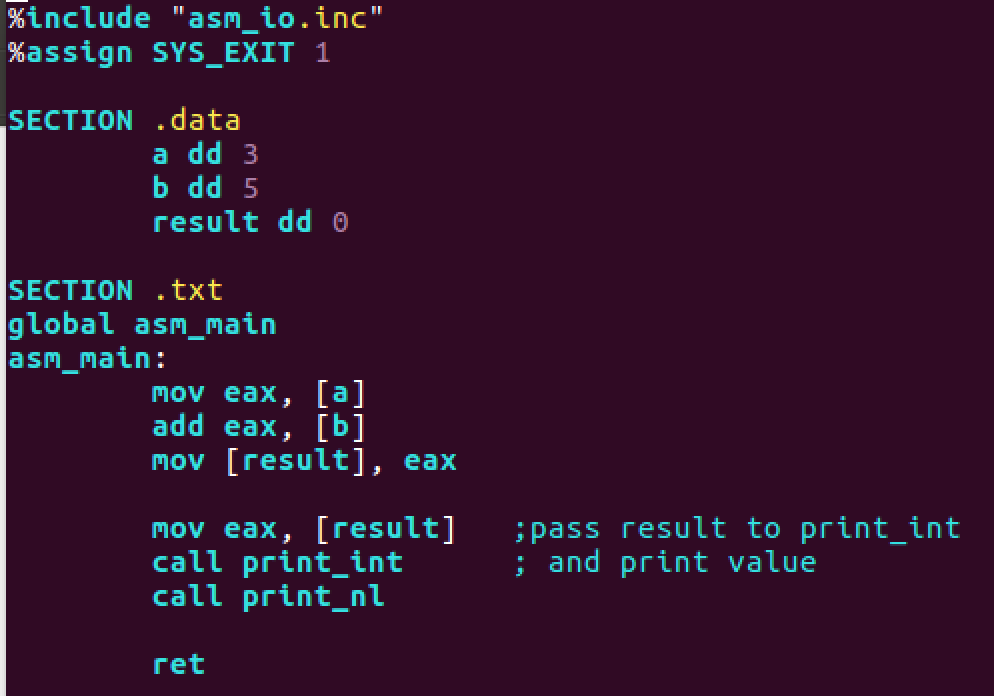


ex4.asm (64-bit function)

To compile, link and run:

* nasm –f elf64 ex3.asm
* nasm –f elf64 ex4.asm
* gcc (–fno-pie -no-pie) driver.c ex3.o ex4.o
* ./a.out

### Create functions that uses print library (asm\_io.asm, and asm\_io.inc provided).



ex5.asm

Copy the provided (D2L) asm\_io.asm and asm\_io.inc

For everything to work in –m32 you may need to first install this:

sudo apt-get install gcc-multilib

To compile, link and run:

* nasm –f elf32 –d ELF\_TYPE asm\_io.asm
* nasm –f elf32 –d ELF\_TYPE ex5.asm
* gcc –o prog2 –m32 driver2.c ex5.o asm\_io.o
* ./prog2

Notes:

For gcc, use **-m32** if you compile 32-bit files on 64-bit machine, generates 32-bit i386 code

For nasm use -**f elf32 –d ELF\_TYPE** to map \_printf to printf calls.

For gcc, use **-fno-pie -no-pie** for gcc 5.0 and up (something to do with not being able to use 32-bit absolute address in assembly) <https://stackoverflow.com/questions/43367427/32-bit-absolute-addresses-no-longer-allowed-in-x86-64-linux>