

Introduction to Intel x86-64 Assembly, Architecture, Applications, & Alliteration

Xeno Kovah – 2014-2015
xeno@legbacore.com

All materials is licensed under a Creative Commons “Share Alike” license.

- <http://creativecommons.org/licenses/by-sa/3.0/>

You are free:



to **Share** — to copy, distribute and transmit the work



to **Remix** — to adapt the work

Under the following conditions:



Attribution — You must attribute the work in the manner specified by the author or licensor (but not in any way that suggests that they endorse you or your use of the work).



Share Alike — If you alter, transform, or build upon this work, you may distribute the resulting work only under the same, similar or a compatible license.

Attribution condition: You must indicate that derivative work
"Is derived from Xeno Kovah's 'Intro x86-64' class, available at <http://OpenSecurityTraining.info/IntroX86-64.html>"

Attribution condition: You must indicate that derivative work

"Is derived from Xeno Kovah's 'Intro x86-64' class, available at <http://OpenSecurityTraining.info/IntroX86-64.html>"

Intel vs. AT&T Syntax

- Intel: Destination <- Source(s)
 - Windows. Think algebra or C: $y = 2x + 1$;
 - `mov rbp, rsp`
 - `add rsp, 0x14 ; (rsp = rsp + 0x14)`
- AT&T: Source(s) -> Destination
 - *nix/GNU. Think elementary school: $1 + 2 = 3$
 - `mov %rsp, %rbp`
 - `add $0x14,%rsp`
 - So registers get a % prefix and immediates get a \$
- My classes will use Intel syntax except in this section
- But it's important to know both, so you can read documents in either format.

Intel vs AT&T Syntax 2

- For instructions which can operate on different sizes, the mnemonic will have an indicator of the size.
 - movb - operates on bytes
 - movw - operates on word (2 bytes)
 - movl - operates on “long” (dword) (4 bytes)
 - mov/movq - operates on “quad word” (qword) (8 bytes)
- Intel indicates size with things like “mov *dword ptr* [rax], but it’s not in the actual mnemonic of the instruction
- Will occasionally see things like “movzwl” which is move with zero extend from a word to a long

Intel vs AT&T Syntax 3

- In my opinion the hardest-to-read difference is for r/m32 values
- For intel it's expressed as
`[base + index*scale + disp]`
- For AT&T it's expressed as
`disp(base, index, scale)`
- Examples:
 - `call QWORD PTR [rbx+rsi*4-0xe8]`
 - `callq *-0xe8(%rbx,%rsi,4)`

 - `mov rax, QWORD PTR [rbp+0x8]`
 - `movq 0x8(%rbp), %rax`

 - `lea rax, [rbx-0xe8]`
 - `leaq -0xe8(%rbx), %rax`

And some versions of the gnu tools, instead of using like “mov -0x4(%rbp)” will show it as “mov 0xFFFFFFF4(%rbp)”
and some will use “movl -0x4(%rbp), %eax” and some will use “movl 0xFFFFFFF4(%rbp), %eax”
http://sig9.com/articles/att-syntax-assembly_language

TODO

- Create a game that shows two instructions in AT&T syntax and Intel syntax, and asks the students whether they're the same or not
- (The +100/-200 helps mitigate advantage of guessing)