CSCI 2330 – x86-64 Operand & Addressing Exercises

1. What's **wrong** with each of the following (invalid) x86-64 instructions?

(a) `movq %rax, $0x123`

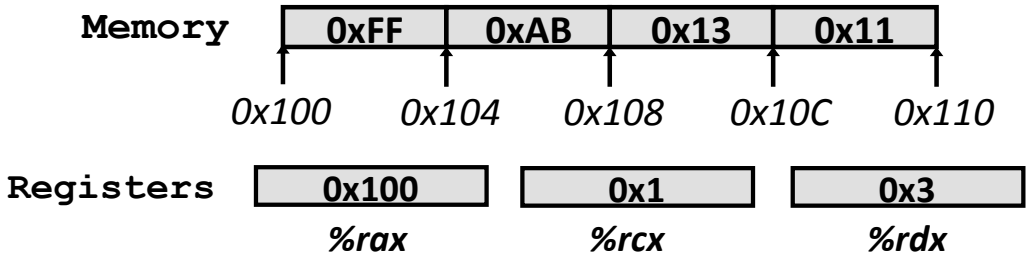
(b) `movq %eax, %rdx`

(c) `movl %eax, (%edx)`

(d) `movl %rax, (%rsp)`

(e) `movw (%rax), 4(%rsp)`

2. Consider an x86-64 machine with the memory contents and register values specified below. For each x86-64 operand in the list below, compute the value that the operand evaluates to. Assume a 4-byte operand size in all cases.



(a) `%rax`

(b) `0x104`

(c) `$0x108`

(d) `(%rax)`

(e) `4(%rax)`

(f) `9(%rax,%rdx)`

(g) `260(%rcx,%rdx)`

(h) `0xFC(,%rcx,4)`

(i) `(%rax,%rdx,4)`

3. For each of the following x86-64 instructions, rewrite as a C-style command using assignment (=), pointer dereferencing (*), and regular arithmetic operators (+, –, etc). You can use register names as subexpressions; e.g., the instruction "`movq %rax, %rcx`" could be rewritten as "`rcx = rax`". Assume there is no data size scaling when using pointer arithmetic.

(a) `addq %rax, %rcx`

(b) `subq %rax, (%rcx)`

(c) `movq 20(%rax), %rcx`

(d) `leaq 20(%rax), %rcx`

(e) `addq 20(%rax), %rcx`

(f) `leaq 9(%rax,%rdx,2), %rbx`

(g) `subq 9(%rax,%rdx,2), %rbx`