CMPSC 210
Principles of Computer Organization
Fall 2015
Bob Roos
`http://cs.allegheny.edu/sites/rroos/cs210f2015`

Quick NASM

This will be *very* quick and *very, very* incomplete! The goal here is to get you writing simple NASM programs for arithmetic expressions as quickly as possible.

For our purposes today, we will say that NASM has four general-purpose registers: `eax`, `ebx`, `ecx`, and `edx`. (The reality is much more complicated!)

Many of the instructions can be used with any combination of register and memory location. For instance,

`mov` `eax,ebx` means "copy ebx into eax" or "eax = ebx"
`mov` `eax,[a]` means "copy memory location a into eax"
`mov` `[b],ebx` means "copy ebx into memory location [b]"
`mov` `ecx,37` measn "copy the constant 37 into ecx"

Most of the commands take two arguments, rather than 3. The first argument is both operand and destination. For instance,

`add` `eax,ebx` means "eax = eax + ebx"
`or` `ebx,ecx` means "ebx = ebx | ecx" (bitwise "or")
`sub` `ebx,1` means "ebx = ebx - 1"
`add` `eax,[x]` means "eax = eax + memory location x"

See the sample programs for examples of the "data" and "text" directives. Note that "`dd`" is like "`.word`" and "`resb`" is like "`.space`".

One important fact: strings in NASM programs don't understand "`\n`"—you must use the ASCII value 10 for a newline. Also, strings must be explicitly terminated with the constant "0" (the null character).

Finally, there are no `syscalls` for printing integers, etc. Instead, we use the C library `printf` function. To do this, we must push parameters onto a stack, call the function, and then pop the parameters off the stack. See the programs for examples.

Other questions? Ask!!!