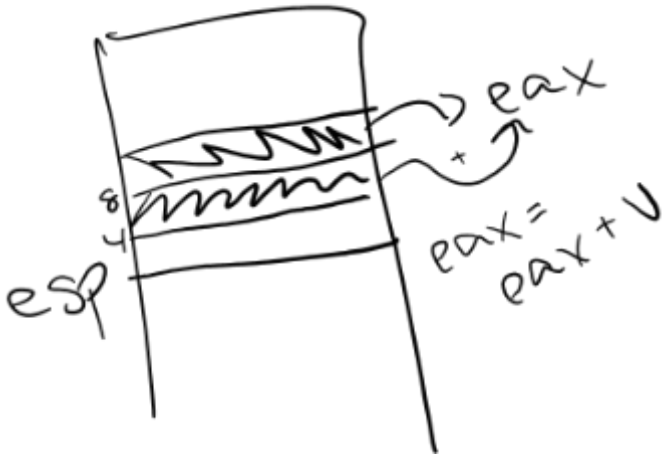
Assembly Language – Arithmetic and Logical Operators

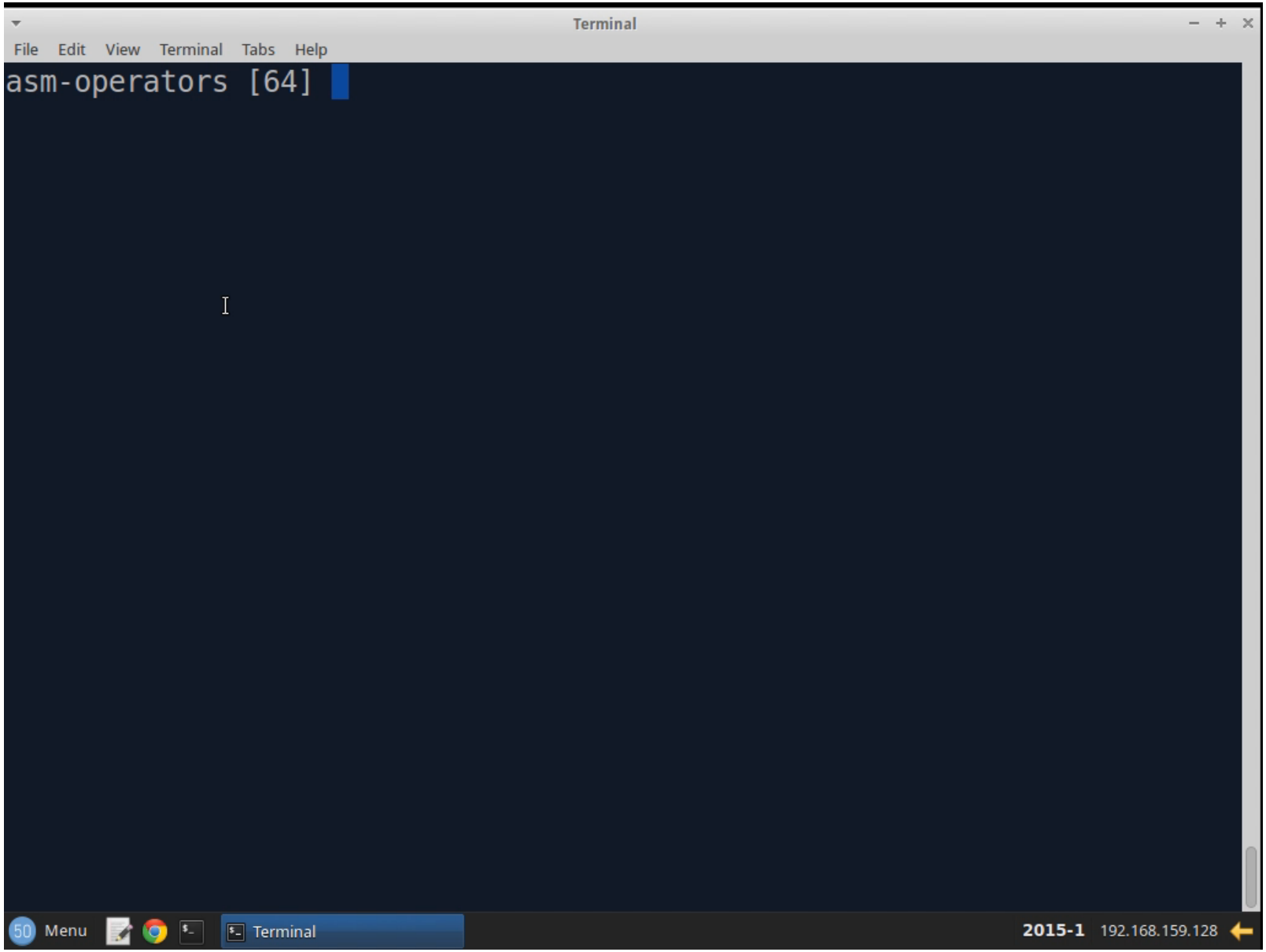
- Learning Objectives
  - Read x86 assembly containing arithmetic and logical operators.
  - Be comfortable read assembly that manipulates data of different sizes.

A Really Simple Function

```
int
sum(int a, int b)
{
    return (a + b);
}
```




```
        .file   "2.c"
        .text
        .globl  sum
        .align  16, 0x90
        .type   sum,@function
sum:
# @sum
# BB#0:
        movl    8(%esp), %eax
        addl    4(%esp), %eax
        ret
.Ltmp0:
        .size   sum, .Ltmp0-sum
```


Terminal
File Edit View Terminal Tabs Help
asm-operators [64]
Menu
Terminal
2015-1 192.168.159.128

Checkpoint 1

- Registers are referenced with `%`
- It appears that you can create addresses using `N(%reg)`.
- Arithmetic operations seem to be of the form:
  - `op2 = op2 OP op1`
- Function parameters are pushed onto the stack in reverse (right to left) order.

Logical Operators

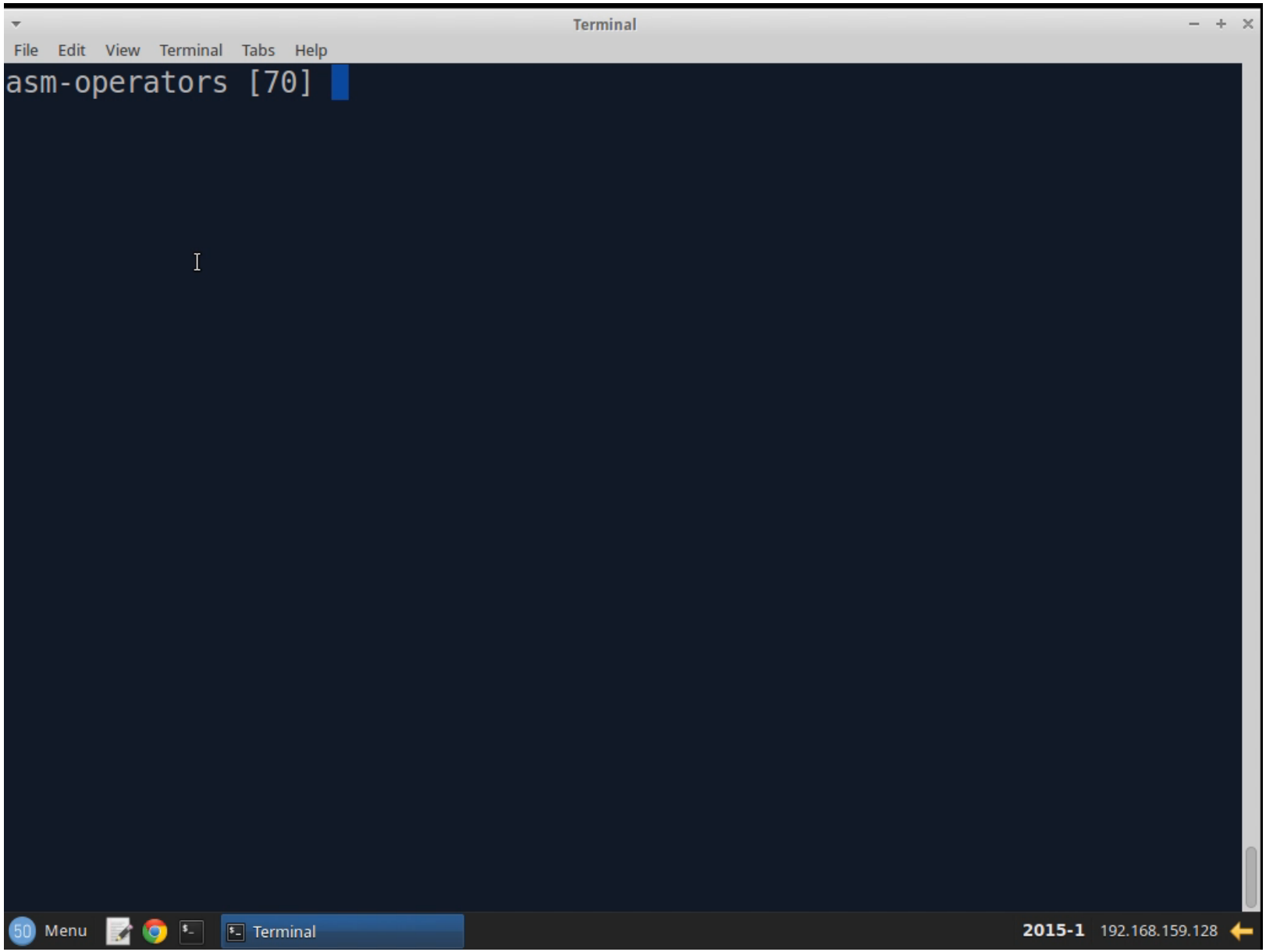
- Recall that you can perform logical operations in C:
  - Given: `unsigned int a, b;`
  - In C:
    - `a & b` produces a bitwise and
    - `a | b` produces a bitwise or
    - `a ^ b` produces a bitwise xor
    - `~a` produces a bitwise not

a & b		
	b=0	b=1
a=0	0	0
a=1	0	1

a | b		
	b=0	b=1
a=0	0	1
a=1	1	1

a ^ b		
	b=0	b=1
a=0	0	1
a=1	1	0

~a	
a=0	1
a=1	0


Terminal
File Edit View Terminal Tabs Help
asm-operators [70]
Menu
Terminal
2015-1 192.168.159.128

Checkpoint 2

- Registers are referenced with `%`
- It appears that you can create addresses using `N(%reg)`.
- Arithmetic operations seem to be of the form:
  - `op2 = op2 OP op1`
- Function parameters are pushed onto the stack in reverse (right to left) order.
- Logical operations work just like arithmetic ones.

Moving Stuff Around

- We've already seen one instruction that lets us move stuff around:
  - `movl 4(%esp), %eax`
- Move instructions are how you copy variables (values) around.
- They are of the form:
  - `movl src, dest`
- You can move things to/from memory or between registers.

Checkpoint 3

- Registers are referenced with `%`
- It appears that you can create addresses using `N(%reg)`.
- Arithmetic operations seem to be of the form:
  - `op2 = op2 OP op1`
- Function parameters are pushed onto the stack in reverse (right to left) order.
- Logical operations work just like arithmetic ones.
- Move instructions move data between registers and memory and between registers. Format is:
  - `mov src, dst`