# CMSC 216
## Introduction to Computer Systems

Akilesh Praveen
Dr. Ilchul Yoon • Spring 2020 • University of Maryland
http://cs.umd.edu/class/spring2020/cmsc216/

---

Last Revision: January 30, 2020

## Table of Contents



---

# 1 Notes

This is a compilation of my notes for CMSC216 as a TA for the Spring 2020 offering of the course at the University of Maryland. All content covered in these notes was created by Dr. Ilchul Yoon and Dr. A.U. Shankar at the University of Maryland, and these are simply my transcriptions of the content provided in lecture and lab, along with my additional insight from the course compiled in a LaTeXdocument.

The notes template in use is Alex Reustle's template, which can be found on his github at the following location: `https://github.com/Areustle/CMSC351SP2016FLN`

Please send errors to `apraveen@cs.umd.edu`

# 2 Introduction to CMSC216

CMSC216 is where you learn how a computer works on a much lower level than you've experienced before. There are 3 main components that the course will explore.

## Overview

- **UNIX** Threads, processes, and pipes as the building blocks of much bigger applications. We will be working with the UNIX operating system on the development environment at `grace.umd.edu`
- **C** is a high-performance language that works at a much lower level than Java. Things like memory management and advanced data structures are left up to the user. We'll cover concepts like memory management, pointers, and system calls.
- **Assembly** is even lower-level than C, and studying it will reveal how processors process instructions, store data, and maintain a stack and a heap. It's the lowest level you'll go in this class. For this semester's 216, you will be using MIPS assembly.

## Debugging

Debugging is accomplished using two main tools: **Valgrind** for memory related issues, and **gdb** for general debugging. Both of these tools are very powerful if used correctly, and are explained as the semester progresses.
For more information, check out this link provided in the course slides:

`The Essentials of Debugging`

## Useful UNIX Commands

Although the UNIX environment may seem confusing at first, learning it is essential to navigating the Grace environment. Below are some of the basic commands that you may find useful when getting started.

- **`ssh`** → If you are not using MobaXTerm, you will have to access grace using the `ssh` command. For the purpose of logging in for CMSC216, I recommend adding the `-y` flag in order to bypass the warning it will give you. E.g. `ssh -y yourdirectoryid@grace.umd.edu`
- **`ls`** → The `ls` command lists all the files in your current directory. You can use the `-l` flag to get more detailed information. E.g. `ls`, `ls -l`
- **`cd`** → The `cd` command changes the directory you're currently in, mainly to directories that you can see with `ls`. Typing `cd ..` will navigate one directory 'up' from your current directory, and `cd` without anything else will return you to your home directory. E.g. `cd 216public`
- **`pwd`** → This command displays your current directory. Useful for finding out where exactly you are in the UNIX file hierarchy. E.g. `pwd`
- **`rm`** → This command stands for 'remove'. It can be used to remove singular files, or can alternatively be used with the `-r` flag to recursively remove directories. E.g. `rm hello.c`, `rm -r project1` (project1 would be a folder.

# 3 Machine

A computer is composed of several parts, but a great way to think about it is a few main components connected by a **bus**.

- **Memory** can just be thought of as a contiguous array of bytes. At the end of the day, this is the stuff that has to be written to/read from.
- **I/O Devices** are connected to the CPU via a bus, like mentioned above. By performing read/write operations to the right adaptor, the CPU is able to interface with different I/O devices.
- **CPU** is the central processing unit of the computer. It handles computational operations (arithmetic, logic, etc.) and interfaces with the memory and I/O devices via the bus. The CPU is also responsible for performing the **fetch-execute cycle**.

A bus is like one main connector that's responsible for making sure the CPU, memory, and I/O devices are all able to interface with each other.

# 4 A Main Section (Template)

Here are some cool notes that will no doubt be very helpful to students.

Here is some cool code in Maryland colors!

```
int main() {
  int x = 1;
  int y = 2;
  printf("hello world");
  return x + y;
}
```

## A Subsection

A subheading's text