Note: there isn't much content here. Read this for a general overview of the course and some of my little notes on each part. I would recommend reading the first two sectionss, though.

Why is this Course Important?

This course is titled "Great Ideas in Computer Architecture" for a reason. All of these topics relate to the fundamentals of how our computers operate. We learn about the memory hierarchy. We learn about how our computers translate text into machine-readable code and how this code is executed. We learn about processes to speed up the execution of our code. While these topics may not be directly relevant to everyone (especially to the "I don't want to be here, but it's a requirement" people—you know who you are), there is definitely merit to taking this class. It's one thing to understand an equation or formula, but knowing its derivation gives each term a new meaning. With this new fundamental understanding, your view of how "code works" will forever change. While you won't always refer to an equation's derivation to use it, having this understanding brings so much more depth into the picture. It's also just really cool to learn about this stuff—at least when you aren't being tested on it, heh.

How to do Well

The best way you can do well in any course is to enjoy the material. With tests and a grade on the line, it's very difficult to do this; this means you should at least (temporarily) convince yourself that you enjoy the material. It's much easier (maybe less painful is a better word) to study for a subject you are interested in, and you may end up enjoying the subject later on. Looking at the long list of topics below, you really should devote a lot of time to understanding the material. It goes by very fast during a normal semester, and summer will be even faster. If you want to do well, you truly have to put the effort in and immerse yourself in this course. There are no shortcuts.

Jon's take on CS 61C

We cover *a lot* of material in this class, so it's easy to become overwhelmed and lose sight of what this class is actually trying to teach. Below is a list of all the topics we will be covering this semester, with short comments on each one. I have included when each section should appear in the course, but it is always subject to change. Note that the **final will be cumulative**, so anything related to MT-1 and MT-2 will be fair game. However, when I use the word "final", I am referring to any material after MT-2 that will be included in the final. Likewise, MT-1 and MT-2 refer to material tested in midterms 1 and 2.

(1) **Number Representation (MT-1 and Final):**

In the **MT-1** part of the course, we cover integer representation of numbers. To do this, we introduce Binary, Decimal, and Hexadecimal numbers. We want you to have a fundamental understanding of how to convert numbers from one notation to the other, the limits of each notation, and how to do basic operations (addition, subtraction, etc.) with each one.

In the **final** part of the course, we cover floating point numbers. This now allows us to represent fractions of numbers; however, it also introduces quite a bit of complexity. The intricacies of floating point are quite difficult, so pay close attention when you get to this section.

(2) **C Programming (MT-1):**

Remember, this is not a C course, even if it is titled CS 61C. C programming is quite a small part of the course, but do expect it to play a large role in MT-1 and one of the projects. C is special compared to Java and Python (the languages the other 61-series courses use) because it forces us to manage memory ourselves, and its usage falls nicely in line with the other topics of the course.

(3) **RISC-V (MT-1 and MT-2):**

This is likely the first time you will be working with an assembly language—it was for me. Assembly languages execute in a way that is much closer to how the actual hardware executes. The code you will work with involves a deeper, more fundamental level of thinking, and it may take a while to adjust. Note

that previous iterations of the class (before Fall 2017) used MIPS, a very similar assembly language. There will be slight differences between the two, so be aware of this when doing old exams for practice.

(4) **CALL (MT-1 or MT-2)**

CALL stands for Compiler, Assembler, Linker, and Loader. Each word refers to a different stage in translating programs from text to machine-readable code. **It may seem simple and easy, but don't let that fool you**. I did not spend the time to carefully review each step of the process, and I lost a non-negligible amount of points on these questions. Don't do what I did; don't overestimate your knowledge of CALL.

(5) **Logic (MT-1 or MT-2)**

Logic circuits are quite an important part of the course, and they overlap with the next topic. We learn how to translate written expressions, such as $(A \wedge B) \vee C$ or $(A * B) + C$ depending on notation, into logic circuits and vice versa. The circuits themselves are slightly different from what you have seen in a physics or EE class. You will have registers and operators that now take time (not instantaneous) to complete their execution, and we have to characterize this behavior to ensure *correctness* of our output.

(6) **Datapath (MT-2)**

This is like a big, higher level version of a logic circuit. The Datapath is how our computers execute code (specific to RISC-V), so it takes in an argument, does various circuit-level operations on it, and completes the desired task. You will be implementing this in the logisim project, and we will thoroughly test you on how the datapath works. **Do not** let yourself get carried on the project if you have a partner. **Do not** save studying for the last minute.

(7) **Caches (MT-2)**

Yet another *super* important part of the class. We use caches to speed up memory access, and we will cover everything from types of cache storage to coherency between multiple caches and memory. This is arguably one of the most difficult parts of the class, and there are *lots* of things that could go wrong when attempting these types of problems.

(8) **Performance(MT-2 or final)**

Everything nowadays is about improving performance, and we go over ways to speed up our execution. This includes using multiple processors at the same time without losing correctness, accessing data in efficient manners, and more.

(9) **Virtual Memory (final)**

Virtual memory is truly magical. It allows programs to store memory under the "same" address, which makes memory management significantly easier. It also isolates two programs from each other, so they aren't able to access memory they shouldn't be accessing. The class goes through this part quite fast, and even I am still shaky on the intricacies of Virtual Memory.

(10) **Misc. Topics (final)**

These topics may include dependability (RAID, Error Correcting Codes, etc.), Disks, Warehouse-scale computing, I/O, and more. Also note that Finite State Machines (FSM) appear somewhere in the course, but it doesn't really align with the other topics.

Wow. That was a lot.