Syllabus: CIS 356Assembly Language and Computer Architecture

Brian C. Ladd

Spring 2024

Catalog Description

> `Principles of digital logic, sequential and combinatorial logic circuits, the von Neumann architecture, and microprocessor design. Introduction to binary and hexadecimal number systems, integer representation in binary, assembly and machine language, and subprogram call and return implementation. Prerequisite(s): CIS 203 and CIS 300. Spring`

Overview

Modern computer science education teaches programming using a *high-level* programming languages. Did you ever wonder what Java (or Python, or C++, or ...) was at a *higher-level* **than**? That is, levels of **what** lie below Java and what holds them up at the bottom?

The short answer is that Java lets you program at a high-level of *abstraction*. The (beginning) Java programmer is isolated from the iron and taught to program an abstract, virtual machine that understands and executes **Java**. Everyone then pretends that there is no cost in translating an `if` statement into bits and bytes, the only things a **binary**, **digital**, **general-purpose** computer can actually manipulate.

Actually, it is worse than the above implies. The electronic, physical machine is not hidden beneath a *single* layer of abstraction. Rather there are multiple layers, each defining an interface that abstracts away some of the pesky details of the level below. Using layers to defeat complexity is a well-known (and well-worn) approach in computer science.

This course looks much closer to the hardware, starting with the MIPS *assembly language,* learning to translate it into *machine language*, and then seeing how the resulting bits, loaded into the *instruction decode register* cause electricity to flow through the circuits of the *central processing unit*.

Along the way, the textbook presents the **Big Ideas** of computer science (any wonder that one of them is *abstraction*?) and we examine how numbers, characters, and *instructions* are encoded into **binary** digits. This universal encoding into bits underlies the **general-purpose** nature of the computer.

The circuits of the CPU are built using **digital** logic gates. These gates are the low-level (though still abstracted above the layers of physics) building blocks of a modern computer. Logic begets arithmetic and control, then flip-flop circuits beget memories, and the next thing you know we are talking about a real computation device that can store and execute a *machine language* program. Machine language, zeros and ones, is difficult for humans to program in so we use *assembly language*, a layer that closely follows the machine language but abstracts away the exact bit patterns that drive the circuits.

There are still more layers from assembly up to the high-level language; there is an entire operating system worth of abstraction to be wrapped around the computer studied in this course. But this course covers from digital design of the *hardware* up to the human-readable assembly instructions that define *software*.

Course Materials

Textbook Patterson, D. and Hennessy, J. *Computer Organization and Design: the Hardware/Software Design 5E*. Morgan Kaufman. 2014.

Other Lecture videos, assignments, and additional readings found through the course schedule.

Calendar Considerations

- The last day to **add/drop**: 2024-01-26
- The last day to withdraw or select S/U: 2024-03-29
- **Winter Recess**: 2024-02-22 – 2024-02-23 (days w/o classes)
- **Spring Recess**: 2024-04-01 – 2024-04-05 (days w/o classes)
- **Last Day of Classes**: 2024-05-10
- **Final Exam**: 2024-05-13 12:30 – 14:30
- Additional Calendar Information: SUNY Potsdam Academic Calendar.

Student Learning Outcomes

The *outcomes* of `Computer Organization and Assembly Language` are *specific* and *testable*: these skills will indicate what you have *learned* in the class. Upon completing this course, students should be able to:

1. **Computer**. *Support* the description of a modern computer as being digital, binary, and general-purpose with *concrete reference* to character, integer, and machine instruction encoding schemes. (Apply)
2. **CPU**. *Construct* a non-pipelined, single-clock-cycle, RISC CPU from fundamental synchronous and combinatorial logic gates. (Synthesize)
3. **IDR**. *Trace* the fetch-decode-execute cycle in the computer's CPU and *explain* how the contents of the instruction decode register (IDR) are both hardware and software. (Evaluate)
4. **Programming**. *Implement* any standard CS1 program in MIPS assembly language: structuring control with selection, iteration, subroutines, and recursion; structuring data as integers, chararters, strings, and 1- and 2-dimensional arrays. (Synthesize)

Note: It would be a very good idea to review this section of the syllabus before exams (especially the final). You can also use it when filling out course evaluations to review what the class taught and what it was expected to teach.

Objectives

The *outcomes* address high-level changes in student thinking. Just telling you that is what we want to happen is unlikely to make them happen. The *objectives* in this course are the nuts and bolts of what is "covered" in the course.

The Big Ideas

- Moore's Law
- Abstraction
- Common Case Fast
- Parallelism
- Pipelining
- Prediction
- Memory Hierarchy
- Dependability

Design Principles

- Simplicity favors *regularity*.
- Smaller is *faster*.
- Good design demands *good compromises*.

Computer Architecture

- The Big Three
  1. Central Processing Unit
  2. Memory Hierarchy
  3. Input/Output
- von Neumann's Architecture
- Instruction Decode Register (IDR)

General Purpose Computing

Computer Science Program Student Learning Objectives

The Computer Science department has adopted five **program** learning objectives for students completing any undergraduate major in the department [See https://cs.potsdam.edu for the complete list.] `Competer Organization` addresses PSLOs #2 and #3:
Students graduating from the Computer Science department at SUNY Potsdam are expected to be able to

2. *Solve problems through analysis and implementation of tested programs that use data structures and algorithms.*
3. *Program the multiple layers (e.g., compiler, operating system, network, assembly language) between a high-level programming language and the underlying hardware.*

Assessment

This is a 300-level course in your major field. You will be evaluated on how well you can apply the expected learning outcomes. This means your understanding of *digital logic* and *assembly language*. You will also be graded on how well you can communicate your understanding in English (the language of instruction), formal logic notation, digital logic circuits, and MIPS assembly. Your writing and speaking should be clear, concise, and correct.

Philosophy

I have been working very hard since the plague struck to improve student learning in my courses. The grade (A-F, 0-100, or 0.0-4.0) is a very small amount of information. What **you** have learned across a semester distilled down to less than seven bits.

Much of the work in this class is *practice*. You should set aside time **every day** or so to play with concepts from this class. That is necessary because learning requires multiple sessions engaging with material. You will also need time to read the textbook **outside** the classroom. The *slides* are not enough. You will fail the course if you do not read the book.

One of the goals of lots of practice is for **you** to be able to evaluate your own work. That is, at first, working in groups you will be able to give your classmates constructive criticism on the way to being able to look at your own work critically.

Notice: You (yes, **you**) will make mistakes. Rather than being a sign of some sort of failing, mistakes are a sign of growth. When you make mistakes, you prime you brain for learning about the material and will learn the correct approach much more firmly for having corrected the mistake.

One shortcoming of videos, textbooks, and slides: they are edited to avoid showing incorrect information. This makes it look like a "real" programmer does not make mistakes. Note that the difference between novice and expert programmers is in the *types* of mistakes they make.

Editing, rewriting the messy, perhaps error-prone work that produced the program into a clear, easy to follow result, is the secret. **Hint**: That means you should plan on rewriting your notes, your practice in your notes, and, especially, anything you plan on turning in.

Remember that mistakes as you learn are a sign that you are engaging with the material. They serve as a place to jump off of toward the right answer.

You will be assessed. You will do assessment assignments that you are expected to do on your own *outside of class* to demonstrate what you have mastered. Then, every week or so, there will be a quiz and every several weeks an exam.

In the spirit of mistakes being a *part* of learning, every assessment will be returned with feedback on what you did and did not show mastery of.

The limited amount of time for work is because the rest of the class keeps right on rolling along and there is a lot of material to cover. The longer it is from when you practiced something, the harder the assessment will be and the class will get further and further behind. My cutoff is set to go to zero when I believe your continuing to work on the old material will adversely impact your current learning.

Feedback

`Computer Organization` looks at the lowest level of the machine, built directly atop *logic gates* and *assembly language* is a barely-disguised version of the machine's encoded instructions. They are an important layer lying below every program you will ever write.

Note that learning is an **active** process on the part of the learner (that is you): read and engage with the text and end-of-chapter problems; write program after program after program in MIPS; talk about solving problems and writing programs with your classmates; explain how to solve problems/programs in class; reflect on programming.

The *theme* in the list of your learning activities: **generating** answers. Example solutions (in books or class) give students the wrong idea about solving problems and writing programs. Solutions to real problems are not written down in a clean quarter of a page with no crossing out and no false starts. Real programs are built iteratively, with first language bugs and then logical bugs being worked out. If necessary, the solution can be *rewritten* to look like it was generated without any false starts (just make sure to include enough documentation so that you can understand what you did at some point in the future).

Compare it to an essay: in a book, blog, or newspaper, an article appears, written in correct English with careful word choice, a pleasant cadence, and a strong, logical flow. When *you* write an essay, it does not look like that when you start: research notes on napkins, big X-marks through ~~hole~~ whole paragraphs, and missing topic sentences. The next (and next) draft make it better. You get **good** at writing essays by writing and reading a lot of essays. The same is true about programming: read good solutions and engage with real problems over and over to build up your intuition. This is a **skill** that grows with practice.

Grading

All grades in this course will be on the 0.0 – 4.0 scale. When a penalty is taken, it will be calculated by multiplying the grade by the appropriate fraction and rounding to nearest third, *e.g.* 10% penalty, work graded as a 3.3: recorded grade is 3.0.

Grading Distribution

Category	**Percentage**
Practice	**—No Points—**
Participation	0.10
Group Work	
Homework	0.10
Programs	0.40
Quizzes	0.10
Exams	0.30

Practice Many problems, homework, and worksheets are *practice* for your learning. During class, you are expected to do the problems and be able to explain them when called upon. These may, occasionally, be done in *ad hoc* groups. Remember that the point of these exercises is to **practice** solving problems and **learn** the principles of discrete math.
The most important thing to do when practicing: **Avoid** looking at models, notes, or on-line. Do each problem as if it were a quiz with no notes or book allowed. Recall, if it is successful, is the best way to cement what you know; if unsuccessful, it actually primes the brain to record the answer when you finally look it up. Do *not* give up easily. The harder you work your brain in *practice*, the easier tests and graded assignments will be.
These are practice for a reason: you are learning how to solve the problems. Making mistakes is *encouraged*. These are *formative* assessment, assessment that helps form your knowledge. Learn from your mistakes.
Participation How do *you* learn things that are important to you? A commitment to sustained, disciplined practice is part of it for most things, including both mental and motor skills.
I **invite** you to make such a commitment to `Computer Organization`, agreeing to: read the textbook; work through examples and practice; join the class on time, every time and be part of the learning community.
Practice and example problems are digital logic and assembly language *exercises*. Just like running or strength training for sport, mental exercises, done regularly over time, are necessary to develop new skills.
To check the preparation and encourage participation, I will have randomized class roster (that will change when I feel like it) and will direct my questions to specific students during class. Additionally, students will be asked to present their solutions to the problems we are working in class.
Each week, you will be assigned an integer score 0 – 4 based on your level of participation. At the end of the semester, these points will be averaged and count toward your course grade.
Group Work About once a week you will meet with your assigned group to do a group worksheet during the class meeting time. This is another form of practice with the answers coming from the *consensus* of the group. The point of these exercises is to work together with the other members of your group. Doing these worksheets on your own will not help you as much as talking about the answers and the questions.
The point of these exercises is to work together with the other members of your group: talking about the material, **aloud**, following each other's thought process, reading and writing together. The questions in the group work *are* practice but doing them alone is not nearly as helpful as participating in a discussion about discrete math with your peers. If you get the material easily, helping others has been shown to cement it in your memory; if you struggle with some topic, being able to ask your peers for assistance can be less intimidating than seeing the instructor or tutors.
After each group work session, you will get an e-mail with a short follow-up question. This problem *will* be assessed as part of your participation score.
Homework Occasionally there will be homework problems given. A few will go out at the end of one class to be done in preparation for the next class. Most will be posted to the classroom management system with more questions and about a week to do them.

Programs These will be turned in through GitTea and will be in Java/Psuedocode, MIPS assembly, and Digital circuit files. These will be marked up with feedback.
Quizzes About once a week there will be a quiz. Quizzes will be about the assigned readings, assigned programming practice, or topics covered in lecture. They will account for part of your grade but also serve as feedback to Dr. Ladd as to how the class is learning.
Exams There will two in-class exams at about the thirds of the course and a (doubly-weighted) final at the end. Exams will be closed book and taken without any computer/calculator access.

General Rules

Your choices make your fate. Information arms you to make your best decisions and enjoy the best fate. This section describes the class expectations so that you can meet them (or decide not to).

Instructor Expectations of Students

Communication

Read/respond to e-mail. Read/respond to the Brightspace site. These are the two primary means of communication in the class. You should make use of them.

Brightspace contains the course *schedule*. That schedule contains readings, writings, programs, presentations, and homework deadlines appear. If anything is missing or you find discrepencies, ask.

Students are expected to have a copy of the textbooks (if any) and are expected to complete reading assignments *before* the beginning of class the day they are due. In-class participation requires you to have engaged the reading.

Students are expected to listen actively in class. They are also expected to take notes in class. These two activities (which are linked) correlate strongly with understanding the material presented. Do not copy slides or board notes directly: digest them and restate them in your own words. Mark where you expect test questions to lurk (sometimes the instructor will even help with this).

Advanced courses often use a version control system (such as `git`) to retrieve and submit source code and programming assignments. This requires you to understand how to use `git`, to understand the difference between the *class* repository (which belongs to the instructor) and *assignment* repositories (which belong to you).

Attendance

American undergraduates are old enough to join the army, vote, and even get married. You are each mature enough to make your own decisions about attending class. Be aware that you decisions have *consequences*.

Students are expected to attend every class. Students are responsible for all material covered in every class meeting. There is no taking of attendance in this class (see ‘old enough’ in the previous paragraph).

The study of computer science is cumulative; past experience shows a strong correlation between high absences and low grades.

The registrar schedules the final exams every semester; I am not able to move them and do not give early finals. Take-home final exams are due *before* the end of the scheduled exam period (it *will* be accepted early).

Do Your Own Work

Do your own work. That should go without saying but it appears that for some students it bears repeating.

Code you plan to turn in for an individual grade should not be shared with any other student; you should never look at another student’s project code, not even for ‘guidance.’

I *want* you to discuss assignments and show off results of your playing with programming different things. Discussion at a high-level, even if it includes details on data structures is fine; debugging assistance is also acceptable though the depth of code analysis necessary for debugging skates on thinner ice; directing someone else on exactly what methods a given class should have or the exact format of a programmer-controlled file should take is beyond the pale.

This does *not* apply to group work: in a group project, all members of the team are expected to see and understand all parts of the project. Cross-disciplinary understanding is a highly valued skill in the real world and a major reason for group work in our curriculum.

Turning In Work

Start early, everything takes longer than you think.

As mentioned earlier, reading is to be done *before* class. Homework assignments are also due at the beginning of class so please complete them before class.

Assignments have a due time: a date and a time when they are due. If they are due electronically (they all are), make sure you submit them the expected way by the expected hour.

Make sure you check the spelling and arithmetic in your assignments. It does not help your grade (or my mood) to turn in sloppy work.

Attitude

Learning is not always easy. Learning is not always comfortable. Students must actively engage the material and be able to ask for help. This does not mean look at the problem for fifteen seconds and throw up your hands because it is 'hard.' It does mean beating your head against it for a bit and then asking me for guidance.

Be respectful. Trust me that I actually have a plan and I know where we are going (okay, I don't *always* know exactly where we are going at every moment but I do have a plan). The readings, the assignments, the lectures, they all go together.

Start early! Everything takes longer than you think.

Student Expectations of the Instructor

Communication

You may expect me to state and keep office hours (you know where my office is, right? ~~301 Dunn Hall~~ Potsdam CS Department Discord). My office hours appear on the course schedule. When I am on Discord, in my voice office, without setting the *Do Not Disturb* setting, you may come on in.

You may expect fairly prompt answers to e-mail (next morning is typical).

You may expect confidentiality: I will not discuss your grades with anyone except my departmental colleagues (and then only to make sure that I am doing my job correctly). I will not share work you have done without your permission.

Attendance

You may expect me to be prepared. This means for class every day and it means with assignments, too. You should expect adequate time to complete an assignment after it is published. You may expect some flexibility (within the limitations of getting through the course materials) in due dates.

Assignments

Expect assignments to be clear (correct spelling, correct grammar). Expect most sample code to compile and run. Expect assignments to be accompanied by a clear system by which they will be evaluated. Expect me to follow that system.

Students should expect that their work will be evaluated on its own contained merits and not based on how closely it follows the instructor's personal/political views (unless I put parroting my political views into the rubric).

Student Expectations of One Another

Wait, students have expectations of one another? Yes, they should. The most important expectation we should all have of each other is to create a safe learning environment. By safe I mean we should all feel safe to ask questions, to admit mistakes, to try and to be wrong.

Remember that trying prepares your mind for learning. Mistakes that you work through lead to deeper learning and understanding of the process.

Students should be willing to help one another and *must* all be aware of the limits.

In group work, all students in the group should expect equitable division of labor and should expect to evaluate every member of the group.

Students should expect to evaluate other students' work (and to be evaluated by other students). Both parts of this process have valuable learning outcomes.

Students should expect to feel safe being different. What does this mean in computer science? I actually am not sure. It means respecting each other, evaluating each other in openly communicated terms related to computer science (since that is the topic of the course), and being flexible, accommodating each other whenever possible.

Grading Grievances

When graded work is returned you may have questions about how it was graded. If the question pertains to process (*e.g.,* addition, number of points per subsection), by all means ask me as soon as possible. Typically, if the grade is changed, you will be asked to e-mail me the grade update so that the grade book is current.

If, however, you feel that your grade fails to reflect the quality of your work (you want to argue that your answer is, in fact, correct, rather than incorrect), take a deep breath and wait. You must wait no less than one and no more than seven days after the work is returned to you to raise the issue with me.

After the cooling-off period, resubmit the work along with a *written* explanation of your concern. This is to assure that your concern gets a cool, thoughtful consideration. If I agree to regrade the work, I will regrade the entire project, replacing the old grade with the new one.

SUNY Potsdam Department of Computer Science Code of Professional Conduct

All members of the Potsdam Computer Science community are governed by the ACM Code of Ethics and Professional Conduct, `https://www.acm.org/about-acm/acm-code-of-ethics-and-professional-conduct`, which we have distilled into our SUNY Potsdam Department of Computer Science Code of Professional Conduct (see below). The department faculty are committed to modeling and promoting ethical and professional behavior for all our students.

Preamble

All members of the ACM, including the Computer Science faculty of SUNY Potsdam, are committed to ethical professional conduct as specified in the ACM Code of Ethics and Professional Conduct. Students, taking courses from the faculty, are bound by our commitment.

All members of the Department are obliged to remind one another to behave professionally. Violations should be reported promptly; however, capricious or malicious reporting of violations is, itself, a violation. When reporting, bring all relevant aspects of the incident to the faculty's attention.

Moral Imperatives

As a Computer Science student I will...

Respect all members of the Department.

1. Be professional in face-to-face and electronic interactions.
2. Be fair so everyone is free to work and learn.
3. Be active in preventing discrimination in physical and electronic spaces frequented by Department members.

Accept and provide appropriate feedback.

1. Avoid starting or spreading rumors.
2. Respect confidentiality.

Be honest, trustworthy, and respect intellectual property.

1. Only take credit for my own work.
2. Respect the privacy of others.
3. Access computing resources only when authorized and report any access risks discovered.

Contribute to society and human well-being.

1. Improve public understanding of computing and its consequences.
2. Consider both the direct and indirect impacts of my actions.

Student Support

Caring Community

I recognize that this is an incredibly stressful time for you, your peers, and our community. Please know that there are resources available to you, both on and off campus, to support you during these very uncertain times. Our excellent Counseling Center staff are available to meet with you; more information can be found on their FAQ page accessed at:
https://www.potsdam.edu/studentlife/wellness/counseling-center/coping-covid-19-pandemic/counseling-center-faqs.
In addition, information on a variety of on- and off-campus resources can be found our Bear Care site:
https://www.potsdam.edu/studentlife/wellness/bear-care. You are an incredibly important member of our Potsdam community; please take care of yourself, and each other.

Every student in this class is a valued individual. If you are struggling with issues outside of the classroom, please know that there are professionals both on and off campus who can assist you. If you need immediate assistance, please contact our campus Counseling Center (with free counseling) at (315) 267-2330 or visit their website. Links to other resources are provided below:

Andrea Waters, Title IX Support Staff & Title IX Core Team

- Sisson 244. (315) 267-2350
- http://www.potsdam.edu/offices/hr/titleix

Bias Incident Reporting

- http://www.potsdam.edu/about/diversity/biasincident

Center for Diversity

- 223 Sisson Hall
- (315) 267-2184
- http://www.potsdam.edu/studentlife/diversity

University Police

- Van Housen Extension
- (315) 267-2222 (number for non-emergencies; for an emergency please dial 911)

Student Conduct and Community Standards

- 208 Barrington Student Union
- http://www.potsdam.edu/studentlife/studentconduct/codeofconduct

Reachout (24-hour crisis hotline)

- (315) 265-2422

Renewal House (for victims of domestic violence)

- SUNY Potsdam Campus Office: Sisson 217 (open Wednesdays, 9-5:00)
- (315) 379-9845 (24-hour crisis hotline)
- Email: renewalhouse@verizon.net

And please: if you see something, say something. If you see that someone that you care about is struggling, please encourage them to seek help. If they are unwilling to do so, Care Enough to Call has guidelines on whom to contact. Everyone has the responsibility of creating a college climate of compassion.