Topic	ID	**Learning Goals Students Can…**
ALU/Registers/ Memory	A1	Describe a basic computer with basic components (ALU, Registers, Memory) and explain how instructions execute and data flows.
Machine Level Instructions	B1	Trace execution of a simple C program and translate to a set of machine level instructions to emulate that C program
	B2	Identify and group Gold Assembly instructions based on their utility for programming(control flow of execution, access memory, arithmetic operations, etc.)
	B3	Describe in what ways instructions and data are the same at the bit level.
	B4	Translate a Gold Assembly instruction into machine representation (in bits)
	B5	Decipher according to Gold Assembly language rules the various parts of an instruction (opcode, operands, etc) from the bit
	B6	Identify what information is available to an instruction statically and what must be calculated dynamically at run time. For example, instructions are created ahead of time and live in memory and are static but that the data they access, including the memory addresses to be accessed may be only calculated or available at run time
	B7	Recognize that subtracting a number from another involves taking the twos complement of the number and adding it. Be able to apply the principles of twos complement to be able to correctly implement sign extension.*
ISA Design	C1	Describe the minimal set of addressing modes needed for an instruction set to be complete.
	C2	Compare and contrast various addressing modes (e.g. the limitations of not supporting a particular mode in an instruction set, why dynamically generated addressing is necessary).
	C3	Compare and contrast the performance impact of addressing modes -- specifically be able to discuss the design trade offs in instruction size, memory versus register access, and direct versus indirect addressing.
	C4	Evaluate tradeoffs in instruction set design. This involves discussion of minimalness, orthogonality, and simplicity, and performance. This should be done for pairs of instructions up to the point of evaluating the differences in CISC and RISC instruction sets.
Variables	D1	Describe the differences between dynamic and static variables in terms of what the compiler can do for each in creating assembly instructions.
	D2	Give examples of both dynamic and static variables in both Java and C
	D3	State for different kinds of variables what information is statically known and what information is dynamically known.
Flow of Control	E1	Keep track of program counter when code using control flow (jumps) is executed
	E2	Calculate jump targets based on the address of the program counter.
	E3	Explain why conditional control flow (loops) is needed enable static programs to compute dynamically sized results.
	E4	Compare and contrast scenarios which require static versus dynamic jump targets.
	E5	Give C or Java code examples which require direct versus indirect jumps and vice versa
	E6	Describe how performance can be affected by dynamic jumps (e.g. be able to show how you can use jump tables to make switch statements faster)
Language Design and Tradeoffs	F1	Explain why procedure return in C/Java must be dynamic – consider the case of a programming language whose procedure RETURN was a static jump
	F2	Explain the consequences to programming if local variables were allocated statically
	F3	Explain the consequences to programming of eliminating dynamically allocated local variables and/or dynamic return.
	F4	Explain the advantage of using the stack for local variables as opposed to just using the heap, including describing how the stack is not required (e.g. you can just have a heap – and that having the stack is a design tradeoff).
	F5	Show how procedure call implementation is different if you use the heap instead of the stack.
	F6	[Understand advantage of maintaining a closure after a procedure returns and that this would require using the heap instead of the stack. Advanced students only
	F7	Show the machine instructions necessary to implement a procedure call and return and describe the format of the stack
	F8	Explain why a procedure-calling convention exists and the design tradeoffs of having it implemented by the compiler and not imbedded in the instruction architecture alone
	F9	Explain how the independence of callers and callees complicates the planning of register usage (e.g. what values to store in register). For example, describe how storing all values in the caller is rarely optimal.
	F10	[Develop a heuristic that a compiler could use to determine when to use a callee-save register and alternatively when to use a caller-save register by giving examples in machine code that benefit from each choice.]
External	G1	Explain what PIO and DMA are and how they differ and are similar to each other
~~Devices and Files~~	~~H1~~	~~Explain what disk drive characteristics contribute to how quickly information can be retrieved from disk~~
	~~H2~~	~~Calculate average disk access time~~
	~~H3~~	~~Explain how sectors are identified (head, track sector)~~
	~~H4~~	~~Explain and compare the tradeoffs disk scheduling algorithms make~~
	~~H5~~	~~Describe and draw pictures of the UNIX file system, basic building blocks and on disk data structures including blocks, inodes, and files~~
	~~H7~~	~~Apply knowledge about disk performance characteristics to data layout on disk~~
	~~H8~~	~~Explain how failure of the OS impacts various structures in the file system – at various points of time of failure, depending on the status of the write in a file system. (this will likely be going away).~~
~~Networking~~	~~I1~~	~~Compare and contrast the communication model for procedures on a single machine (the procedure call model) versus networked communication (these differences include: make a connection, transfer data, shut down the connection).~~
	~~I2~~	~~Write a simple networked program (e.g. perhaps a very simple web server involving a client and server getting connected), including gaining familiarity with networking APIs.~~
	~~I3~~	~~Describe how networked communication follows an asynchronous communication model in which synchronization needs to be handled explicitly~~
	~~I4~~	~~Describe how sending a stream of data across a network involves chopping that stream into chunks, sending them independently, chunks can get lost, and that reliability issues arise and must be dealt with. Describe the role that a protocol plays in abstracting these issues.~~
	~~I5~~	~~[Protocol stack and layering (design, layers of abstraction)not covered in 213~~
Processes	J1	Explain that there is a private address space for each process and that that hardware does the translation (via base-bounds).
	J2	Explain the design tradeoffs of why virtual addressing is needed and desirable and also the complicating and performance implications.
	J3	Explain that processes are separate entities with their own address space and that if two processes access the same address location it's different and that this is an example of virtual memory.
	J4	Describe a motivation for processes based on an example of why we need to move from asynchronous access to concurrent access with synchronization primitives
	J5	[Describe at a basic level the tradeoffs (via analysis with examples of round robin, the role of the kernel clock, pre-emption, interrupts) available for scheduling of processes.]
	J6	Trace though code with a producer/consumer relationship.
	J7	Use synchronization primitives to enable mutual exclusion access, e.g using semaphores to control access to a shared array.
	J8	Use synchronization primitives to enable signaling in producer/consumer structures
	J9	Explain how threads and processes differ specifically with regards to shared memory
	J10	Describe real world scenarios which require the use of concurrency via multi-threading
	J11	[Would like: Explain how processes converts asynchrony into concurrency by using synchronization primitives. Interrupts.
	J12	Compare and contrast the synchronization features that students already know from Java with the variations available in C and Unix
	J13	Compare and contrast the threading features that students already know from Java with the variations available in C and Unix
	J14	[Explain how when a program has more than one lock, that it introduces the possibility of deadlock. Give an example of a code that has the possibility of deadlocking and a different example with live-locking. Explain the tradeoffs associated with different granularities of locking. Can explain the standard dining philosophers' problem. Priority inversion and techniques for dealing with it?]
Java and C comparable	K1	Write C code equivalent to known Java code (for the subset of C that is basically the same in both languages – primarily the imperative structures and primitive types
	K2	Describe how arrays are different in C and Java (C arrays are static and Java arrays are dynamic)
	K3	Use C syntax for pointers and compare that to reference variable use in Java.
	K4	Describe the similarities and differences between C structs and Java objects and specifically how their features are addressed in assembly code
	K5	Do pointer arithmetic in C.
	K6	Describe that dynamic memory allocation is the same in C and Java but that type safety is different
	K7	Describe how memory reclamation is different and be able to write C programs that use memory reclamation
	K8	Describe how garbage collectors only solve one of these two memory problems: dangling pointers and memory leaks (including being able to describe these two problems and give code examples which would create them).
	K9	Create a jump table to implement a C switch statement.
	K10	Describe why Java's polymorphism required indirect jumps and discuss the performance implications of that
		Read and understand basic C programs.