**General Guidance:**

Go over the “Topics Covered By Day” in Moodle, and make sure you know it all. Go over the homeworks and quizzes and know the solutions, which you can rewatch online in Moodle using the links to the videos. (these links are associated with the dates in there – and also in all the emails that I have sent about class recaps)

* Digital Logic
* truth tables
* determining number of rows, number of columns, inputs versus outputs
* sum of minterms
* sum of products
* minimum sum of products
* k-maps, 2-, 3-, and 4- variable
* know how to use them
* schematics / circuit diagrams
* going from idea to TT, to equation, to circuit, and vice-versa
* basic Boolean algebra
* associative, commutative, distributive, De’Morgan’s Law
* distributive law
* AND over OR and
* OR over AND
* A\*0, A+0, A\*1, A\*0, A+A, A\*A
* mux, demux, encoder, decoder, priority encoder
* importance of the valid bit
* how to use multiple MUXes to build larger MUXes
* XOR and XNOR – relationship to comparators, to overflow detection
* Doing Boolean algebra with this
* ALU as example of combinational logic device
* Implementation and control bits
* Make sure you know the concepts from the homework, etc
* Positional number systems (integers)
* different bases of interest, 2, 4, 8, 10, 16, for example
* converting between bases
* signed versus unsigned, 2’s complement (radix complement)
* ranges of integers that can be represented, total number of numbers that can be represented
* overflow, what does it mean ?
* with unsigned integers base 2
* with signed 2’s complement integers base 2
* sign-extension
* Chapter 2 concepts
* RISC versus CISC
* MIPS assembly programming
* LA, LI, immediate and non-immediate arithmetic and logic operations
* take a high level idea (like in C) and convert to assembly
* take a given MIPS program and figure out what it is doing
* memory operations
* pointers, pointer arithmetic
* lw and sw (and the half-word and byte analogues)
* how the addressing works
* flow control
* if-then
* if-then-else
* complex / compound conditions like if(A<B && C>D)
* using the set / comparison instructions in conjunction with beq and beq to get the job done
* unconditional jumps ( j label)
* conditional jumps (beq Rs, Rt, label)
* loops / iteration
* how and where to put the jumps, and comparisons
* for, while, and do-while
* traversing arrays
* base address of arrays
* pointer computations
* register usages
* a, v, s, t registers
* MIPS instructions
* R, I and J types
* encoding and decoding instructions (if you have the greensheet ref page)
* widths of fields and relationship to number of registers and immediate value widths, relationship to pseudoinstructions for LA, as example
* understanding the program counter
* how to read memory snapshots from CPUlator, and knowing how the addressing words. Bytes, Words, and halfwords.
* Big Idea, stored program computer
* [Von Neumann Architecture](https://en.wikipedia.org/wiki/Von_Neumann_architecture) everything is stored in RAM
* shifting left and right
* relationship to multiplication and division for integers x or / 2 for each BIT
* how to use to get a specific bits (bit manipulation) sll then srl
* bit masking
* & and | (bit wise anding and oring)
* bit-wise NOR and relationship to bit-wise NOT (inversion)
* what is a basic block

**WE HAVE NOT DONE THE BELOW YET THAT MUCH – Keeping here just for reference**

* Chapter 2 concepts
* function calls
* calling conventions
* prolog and epilogs, what to do in them, and how to do them
* pushing and poping
* stack usages
* local variable storage,
* excess parameters
* save/restore RA
* save/restore used registers (like s or t in specific circumstances)
* stack pointers
* stack frame / activation record
* stack versus heap memory usage
* string copy and sort are great examples of leaf and non-leaf functions and function calling taking place, and also for understanding pointers
* MIPS System Calls
* printing
* effect of compiler optimizations on IC, CPI, as an example, and then how this impacts T in the T = IC\*CPI/ F model.
* arrays versus pointers in C
* trend in size of x86 ISA over time
* C Programming
* components of a C program:
* #include’s
* int main(int argc, char\*\* argv)
* what each of these means
* passing parameters to main from commandline
* atoi(), atof(), etc
* usage statements
* declaring and using functions, where the functions have to be declared, function headers
* floats, ints, doubles, char’s, strings,
* pointers
* relationship to addresses in MIPS
* pointer to an item (like an int) versus a pointer to possibly multiple ints in an array (i.e. a pointer versus and array)
* pointer to a pointer
* defererencing a pointer (\*p, \*\*p, etc)
* declaring an automatic array, (array in the stack) versus on the heap
* malloc()
* free()
* file IO
* fread(), fwrite(), fopen(), fclose()
* Endianness
* big endian v little endian (which does x86 use and what about MIPS)?
* hexdump
* Fractional number systems (chapter 3)
* IEEE 754 floating point standard
* converting from base 10 and base 2, fractional, into binary string of IEEE 754, for single precision, primarily
* number of bits for sign, fraction and stored exponent, the bias, how to use it and what it is for
* fixed number systems
* like we did in class
* special cases, +/- INF, +/- NAN, 0.0
* normalized versus denormal numbers
* what does denormal hope to accomplish?
* In the normal case, what is the largest, and smallest numbers possible, base 2?
* catastrophic cancellation
* non-associativity of floating point
* chapter 1 concepts
* 7 great ideas of computer architecture
* definition of “ISA”
* be able to list them
* difference between response time and throughput
* T = IC\*CPI / F, what those terms means, how to calculate
* frequency, period,
* speedup (new scenario versus old scenario)
* when given with times
* when given with throughputs
* average CPI calculations
* Power = C \* V^2 \* f, how trends over time have impacted the terms in this equation
* uniprocessor performance over time, what has happened ?
* power wall
* power versus % utilization of processor
* multi-core ?
* parallelism ?
* Amdahl’s Law