Midterm Study Guide

CS 61C Fall 2019

The major topics covered in this semester's midterm are:

- Number Representation
- C
- Floating Point
- RISC-V Programming
- RISC-V Instruction Formats
- CALL
- SDS, Boolean Logic, and FSMs
- Single-Cycle Datapath
- Single-Cycle Control

We've provided the questions below to help guide your studying of each topic.

Number Representation: Lecture 2

-- Do you understand the differences between the various systems for representing integers in a computer (unsigned, sign and magnitude, biased, one's complement, and two's complement)?
-- What are the ranges of numbers these systems can represent, given n bits?
-- What do numbers represented by these systems look like in binary?
-- Can you easily convert between decimal, hexadecimal, and binary? Do you understand how to convert numbers from decimal to any base-*n* representation?

C: Lecture 3, 4 and 5

-- Can you write a C program for a variety of standard programming tasks with appropriate syntax?
-- Do you understand what a pointer is, how it works under the hood, and when you'd want or need to use it?
-- Do you understand how to dynamically allocate heap memory in a C program using `malloc()`, `calloc()`, and `realloc()`? and prevent memory leaks using `free()`?
-- Do you understand the C memory model (the stack, heap, static, and data segments), and what variables / data go in each?

Floating Point: Lecture 6

-- In a floating point number, what are the significand, mantissa, and sign bit?
-- What is biased notation and how is it used to represent floating point exponents?
-- How do you figure out the step between one floating point number and the next biggest floating point number?
-- How do you figure out the representable range of numbers for a particular floating point representation?

RISC-V Programming: Lecture 7, 8, 9 and 10

-- What is an ISA (instruction set architecture)? How does it differ from a high-level programing language like C or Java?

-- What is a register? How is a register different than physical memory/RAM? How many registers do we have in 32-bit RISC-V?
-- What are the basic RISC-V instructions to perform arithmetic operations on two registers? What are the basic RISC-V instructions to perform arithmetic operations on one register and one immediate?
-- Do you understand why `li t0 0xDEADBEEF` is **not** the same as `lui t0 0xDEADB + addi t0 t0 0xEEF`, and instead expands to `lui t0 0xDEADC + addi t0 t0 0xEEF`?
-- What are the `lw` and `sw` instructions? How do load and store instructions differ from instructions that only change or read from registers?
-- What is big-endian vs. little-endian memory order? How does this affect the results of the `lh`, `lb`, `sh`, and `sb` instructions?
-- How are loops (for loops, while loops) and conditionals (if, else, etc.) implemented in assembly language with branch and jump instructions?
-- Do you understand RISC-V calling convention? Specifically, do you understand the difference between registers that are the callee's responsibility to save, and the caller's responsibility to save?
-- Can you write a function prologue and epilogue and call a subroutine (another function) from within a function? What syntax do we use to call a RISC-V function?
-- Do you understand how recursive functions are compiled to assembly code?
-- Can you read the RTL / Verilog on the green sheet so that if you forget what an instruction does, you can remind yourself?
-- What are pseudoinstructions? Can you convert common pseudoinstructions (`la, ret, mv, li, jr, j`) to the corresponding non-pseudoinstructions? "`jal label`" and "`jalr rs1`" are also pseudoinstructions, even though they aren't defined on the green sheet -- what exactly do they do?

RISC-V Instruction Formats: Lecture 11, 12
-- Which RISC-V instructions fall into each of the six RISC-V instruction types (R-type, I-type, S-type, SB-type, UJ-type and U-type)?
-- What are immediates used for in the five RISC-V instruction types that have immediates (I-type, S-type, SB-type, UJ-type and U-type)?
-- How do you read the "Core Instruction Formats" section of the green sheet, so that you can convert an instruction like `addi a0, x0, 3` to the corresponding 32-bits of binary machine code?
-- What range of immediates can be represented in an I-type instruction, given the number of bits available? What range of immediates can be represented in an S-type instruction, given the number of bits available?
-- What is the program counter (PC)?
-- When not branching or jumping, the default change to the PC for the next instruction is to increment it to PC + 4. What do branch and jump instructions do instead?
-- How does a label get converted to an immediate? What does that immediate represent?
-- Why is the last bit of a SB or UJ type immediate set to 0? How does this affect the range of byte addresses, halfword addresses, and word addresses you can jump to?

CALL (Compilation, Assembly, Linking and Loading): Lecture 13
-- Can you explain what happens during each phase: compilation, assembly, linking, and loading?
-- Do you know what the various parts of an assembled executable are?
-- Do you know how the assembler fills in the symbol and relocation tables?
-- Do you know why the assembler may need to take two passes through a piece of code?

-- Do you know which instructions are pseudo-instructions and which instructions aren't, so you can identify which pseudo-instructions must be expanded by the assembler?

SDS (Synchronous Digital Systems), Boolean Algebra, and FSMs (Finite State Machines): Lecture 14, 15, 16, and 17

SDS

-- Do you know what function the six basic 2-input logic gates (AND, NAND, OR, NOR, XOR, and XNOR) apply to their inputs, and the truth tables that define those functions? Can you draw these six gates, and a NOT gate?
-- Do you understand how an adder is implemented using just basic logic gates? Can you construct an N-bit adder / subtractor from just 1-bit adder / subtractors?
-- What is a multiplexer? How do you implement a multiplexer using just basic logic gates?
-- Do you know what a register and/or flip flop is, and how it is affected by its clock input?
-- Can you explain the fundamental difference between combinational logic elements and state elements?
-- Do you know how to draw and interpret waveform diagrams for circuit systems consisting of both state elements and combinational logic elements?
-- Do you understand what hold time, setup time, clk-to-Q delay, and critical path are? Can you explain what a setup / hold time violation is?
-- Do you understand why hold time <= clk-to-Q delay + shortest CL path?
-- Do you understand why critical path delay = clk-to-Q delay + longest CL path + setup time?
-- Do you understand the relationship between clock period and clock frequency, and how to convert between the two?
-- Do you understand the relationship between the clock period and critical path?

Boolean Algebra

-- There are 9 laws of Boolean algebra that we teach you in CS 61C: can you properly apply each of them to simplify an equation to the minimum number of gates?

FSMs

-- When looking at an FSM, can you identify on the label on top of a transition label which number corresponds to input and which number corresponds to output?
-- Given a bitstream of input, can you follow an FSM to determine an output?
-- In order to determine the output and next state, does an FSM need more information than the input and current state?
-- Given an arbitrary pattern in the stream of bits that you receive, can you design an FSM to detect that pattern, using the minimal number of states?

Single-Cycle CPU Datapath: Lecture 18 and 19
-- How would you define a "CPU"? What is the difference between datapath and control?
-- In a single-cycle RISC-V processor, one instruction is executed for every ____________.
-- What are the five stages of the RISC-V datapath? What takes place in each stage?

-- What is the Register File (RegFile)? What are the inputs into the RegFile? The outputs? Is the RegFile connected to the clock, and if so, why?
-- What is instruction memory? What is stored there?
-- How do you add R-format instructions to the datapath? How do you add I-format instructions to the datapath?
-- What is the program counter (PC)? How does the PC get updated for the next instruction when you execute an `add` or `addi` instruction?
-- What is data memory? What is stored there? Which instructions access data memory?
-- What is the branch comparator? How do you implement branch instructions?
-- How is `jalr` implemented? How is that different from how `jal` is implemented?
-- How do you implement the U-type instructions? (`auipc` and `lui`)

Single-Cycle CPU Controls: Lecture 20
-- How many control signals does the datapath have?
-- What are the values of these control signals for add? Can you determine the correct values of the control signals for any of the RISC-V instructions?
-- Which control signals can be determined in the ID stage? Which control signals must be determined in the EX stage? What signals are inputs into the control logic unit, and what signals are outputs?