What IS and IS NOT in the exam

Digital Design and Computer Architecture
Mohammad Sadrosadati
Frank K. Gürkaynak

http://safari.ethz.ch/ddca

General tips and tricks 1/2

- **The exam will be 70 points of your final grade**
  - Max 30 points will come from the exercises
  - We will have questions that add up to 70 points.
  - Total will be 100
- **We plan to have 1 point == 2 minutes of your time**
  - Exam is 180 minutes, you should be done in 140 ☺, no rush
  - If a question worth 1 point is taking much longer for you, maybe you misunderstood the question, consider skipping it for the moment.
- **Questions generally do not depend on each other**
  - You can solve a question even if you do not solve the previous part
  - If you need the answer of the previous part, write your assumptions

General tips and tricks 2/2

- **Questions generally do not depend on each other**
  - You can solve a question even if you do not solve the previous part
  - If you need the answer of the previous part, write your assumptions.
    "I assume that t_{cycle} is 1.2ns."
- **We have three types of questions. These will be mixed**
  - Minimum you need to know (the questions to get a 4.00)
  - Good understanding (the questions to get a 5.00)
  - Tricky ones (the questions to get a 6.00)
- **There are questions that do not have a RIGHT answer, we want to see your arguments for your choices.**
  - i.e. What is better RISC or CISC, explain why

Introduction

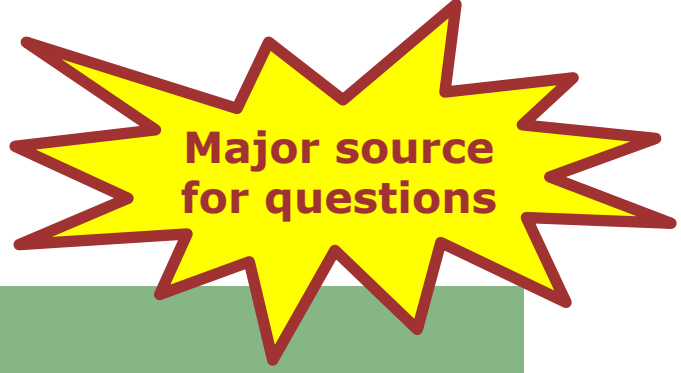
- **You should know**
  - How we handle complexity
  - Why we use abstraction and how it help us
- **Also**
  - That we have a lecture book which is available online

How is Hierarchy, Modularity and Regularity related, how do they help us handle complexity

- **We will NOT ask you**
  - Bio and details of the lecturers

i.e. Which of the following universities was NOT one where Prof. Onur Mutlu spent part of his career

Binary Numbers



- **You should know**
  - How to convert between binary, decimal and hexadecimal numbers
  - Be able to work with numbers that are in powers of 2
  - Be able to interpret numbers as unsigned, sign-magnitude, one's, or two's complement
  - Compare the ranges these interpretations can represent

If the hexadecimal number $(82)_{16}$ is expressing a number in sign-magnitude format, which number does it describe in decimal system?

- **We will NOT ask you**
  - Number formats we have not discussed (i.e. floating point) unless it is covered later in the lecture
    - In the meantime IEEE 32bit FP and IEEE 64bit FP have been covered

i.e. what bit sequence corresponds to 3.1415926

EE Perspective

- **You should know**
  - What a transistor is, its basic function for digital functions, basic materials and components of it.
  - That they can be combined to realize basic logic functions
  - What pMOS, nMOS and CMOS stand for and what their roles are for realizing the functions
  - Basic logic gates: AND, OR, XOR, NAND, NOR, XNOR, inverter and their truth tables

True or false, "pMOS transistors are used in CMOS gates to drive the output of a logic gate to logic low"

- **We will NOT ask you**
  - to CALCULATE electrical properties of a circuit with transistors
  - draw/derive the transistor level schematic of a logic function using transistors

i.e. Draw the transistor level schematic of the CMOS logic gate realizing the function Y= A. (B+C)

Combinational Circuits: Theory

- **You should know**
  - Boolean Algebra, Axioms, Theorms
  - How gates map to basic Boolean operations AND/OR/Negate
  - Rules of combinational circuits
  - Simplifying Boolean logic equations

Which one of the following circuits is NOT a combinational circuit, why

- **We will NOT ask you**
  - Graphical Bubble pushing exercises

i.e. a question like in slide 32-34.

Note that knowing how to push the bubbles could help you with simplification of Boolean logic equations (which we will ASK)

Combinational Circuits: Design

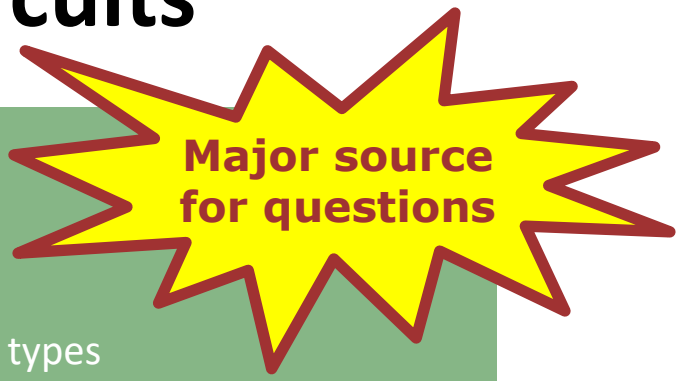
- **You should know**
  - SOP and POS formulations, definitions
  - How to read/construct truth tables, shortcuts with don't cares
  - Karnaugh maps, how to simplify Boolean Logic
  - Contamination and propagation delay

Implement the Boolean logic equation Z=ABC+!A!B+!C!B using ONLY two input AND/OR gates and inverters. On the circuit you have drawn show the critical path, the shortest path by using the delay values for these gates. Simplify this logic function, how much ere you abel to reduce the critical path

- **We will NOT ask you**
  - to design circuits that make use of tri-state logic
  - fixing glitches

i.e. Design a multiplexer using tri-state buffers

Verilog for Combinational Circuits



- **You should know**
  - How to write a Verilog module, instantiate modules
  - Implement basic functions
  - Make assignments between busses of different sizes and types
  - Be able to read/understand and write Verilog code for combinational circuits
  - Draw a circuit diagram from a Verilog code or vice-versa

Which one of the following Verilog statements have an ERROR, explain the issue, and correct the line

- **We will NOT ask you**
  - Implement tri-state busses, circuits using tri-state busses
  - Derive truth tables or work with functions that have Z and X inputs

i.e. How do you write a Verilog code that implements a 32-bit tristate bus with 3 different drivers (Frank, Mohammad, Ataberk).

Combinational Circuits in Processors

- **You should know**
  - Basic building blocks of adders, half-adders, full-adders
  - Ripple carry adder, how it works, the issue with carry propagation
  - Carry-save and carry propagate adders
  - Multipliers and other circuits built from adders
  - Shifters and other circuits
  - ALU and how it operates

Calculate the propagation/contamination delay of a 16-bit Ripple Carry Adder, what is the maximum operating frequency of a processor that uses this adder in its critical path

- **We will NOT ask you**
  - To derive how comparators are obtained from adders
  - Design/draw large multipliers

i.e. Draw the transistor level schematic of the CMOS logic gate realizing the function Y= A. (B+C)

Even more adders

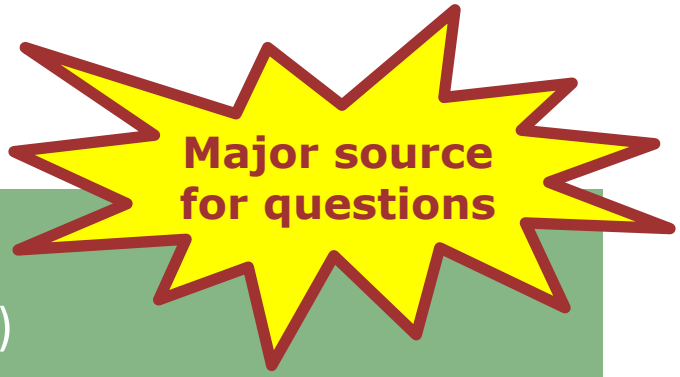
- **You should know**
  - At the moment, we did not yet cover these slides. Maybe later in the lecture we will find the time

No questions

- **We will NOT ask you**
  - What we did not cover in class

Explain different approaches to overcpme the “curse of carry”

Sequential Logic Design



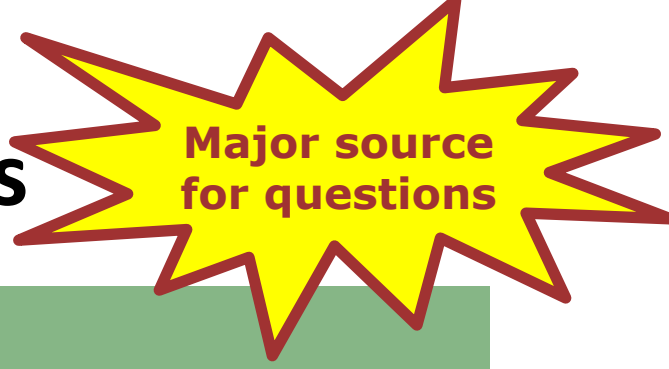
- **You should know**
  - Mealy and Moore (Guaranteed question)
  - Basic state holding elements, latches, FFs, how they work
  - Finite State Machines, how they are constructed
  - Design, simplify, optimize, understand FSMs

Design the FSM that implements "…." derive the logic equations for next state calculation, minimize them

- **We will NOT ask you**
  - To 'factor' FSMs in the exam (i.e. slides 60-63)
  - Draw, transistor level schematics of latches and FFs

i.e. draw the transistor level schematic of a rising-edge trigerred D-type flip-flop with asynchronous reset and enable.

Verilog for Sequential Circuits



- **You should know**
  - `always` statements in Verilog. How they are used
  - How to define/read/understand FSMs, FFs, latches in Verilog
  - If a Verilog description is a combinational or sequential circuit
  - Basic logic gates: AND, OR, XOR, NAND, NOR, XNOR, inverter and their truth tables

Which of the following Verilog descriptions will result in a combinational circuit, and which ones will be a sequential circuit.

- **We will NOT ask you**
  - To evaluate differences/nuances between blocking and unblocking sequential statements (basics of what blocking and unblocking is could be asked)
  - Combinational Verilog descriptions where sensitivity list is manipulated to miss some signals.

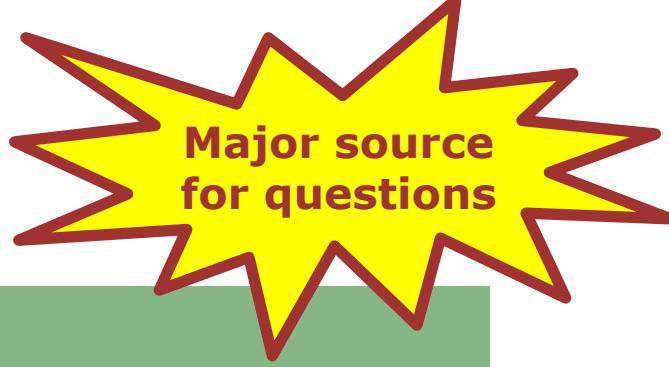
i.e. Show that a combinational circuit described using unblocking statements wi ll work as well as one written with blocking statements

Open Source HW

- **You should know**
  - Nothing in particular from this lecture, it is an informational lecture and not part of the exam

- **We will NOT ask you**
  - Content of this lecture

Sequential Circuits: Timing



- **You should know**
  - Setup and hold time calculations (with and without skew)
  - You need to be up to date on propagation and contamination delays as well

Given a circuit diagram (or Verilog code) calculate if the circuit has a setup or hold violation, if there is a violation explain how it can be fixed.

- **We will NOT ask you**
  - Calculations involving synchronizers (knowing what they do and their basics could be asked, but no detailed calculations on MTBF etc)

i.e. What is the mean time of failure for a circuit running at 1 GHz, when the input changes 100 times a second, Aperture time is 50ps, setup time is 300 ps, and the technology constant is 75 ps

Need for Verification

- **You should know**
  - Why functional verification is difficult
  - How are faults detected (control, activate, propagate, observe)
  - What is a testbench
  - The need for golden models

If you can test 1'000'000'000 input combinations per second, how long would it take you to exhaustively test a 32bit adder.

- **We will NOT ask you**
  - List all possible testbench variations (slide 23)

i.e. Draw three different testbenches and explain what the differences are.

Using Verilog for Testbenches

- **You should know**
  - `initial` statements in Verilog. How/why they are used
  - Defining time in testbenches. `#20`
  - Identify/understand parts of a testbench written in Verilog
  - Helper functions $display, $readmemb
  - The advantages of file-based testbenches

Take a look at the following Verilog testbench. Explain what each section does. Two of the sections has a mistake in the code, identify these, and write the corrected version

- **We will NOT ask you**
  - To write a golden model
  - Discuss the timing of stimuli application and sampling of expected outputs

i.e. Develop a golden model in a language of your preference to implement a golden model for multiplication.

Number Systems

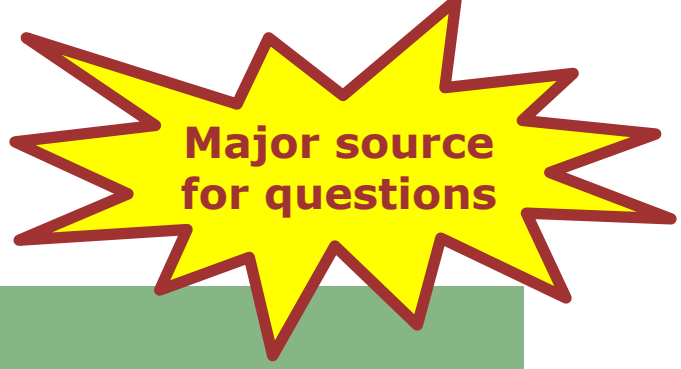
- **You should know**
  - Difference between, integer, fixed-floating point numbers
  - Be able to express simple fractional numbers in fixed-point format
  - Explain the idea behind floating point encoding (sign, mantissa, exponent)
  - Understand different rounding modes
  - The need for special cases (for example NaN)

What is the difference between round-down and round-to-nearest, give an example where the rounding would result in different numbers.

- **We will NOT ask you**
  - To perform a IEEE 32/64 bit floating point number operation using a golden model in binary/hex
  - To identify, describe the Floating Point Unit in page 30
  - To memorize the special case encodings

i.e. Which number 32bit floating point number is represented by 32'h160a_59EF

MIPS Assembly



- **You should know**
  - Essentially MIPS assembly, operands, memory addressing, registers
  - Different types of instructions (R, I, J)
  - Be able to interpret instructions, know what the *basic* instructions we covered in slides in class do (lw, sw, add, addi, bne, j, jal, and, xor..).

Explain what the instruction lw $t0, 24 ($s1) will do. Where is the data coming from and where will it be written to after this instruction.

- **We will NOT ask you**
  - To memorize the registers, all instructions of MIPS. If we ask questions that need such information, these will be provided

i.e. What is register number 17 in MIPS

Memories

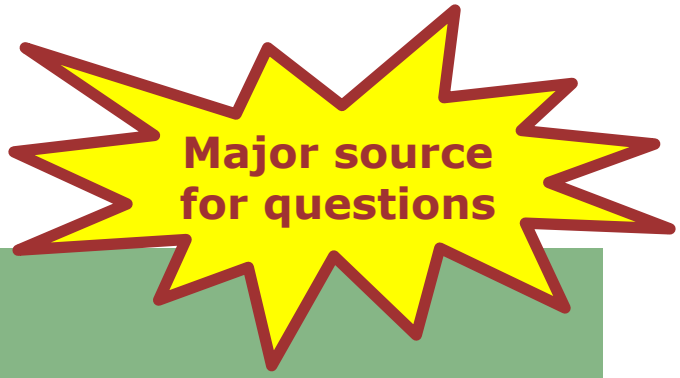
- **You should know**
  - The array structure of memories
  - Types of memories SRAM, DRAM, Register files
  - How larger memories can be constructed out of individual memory blocks

You have an SRAM block that has 8 address bits and can store 32 bits at a time. How many bytes can be stored in this memory. For your 32 bit MIPS processor you need a 4 kByte memory, show a block diagram how this can be constructed using this memory block, connect the address bits and add necessary components.

- **We will NOT ask you**
  - Internal structure of the RAMs using transistor/capacitor and questions related to how electrically the RAM works.

i.e. Explain how the transistors connected to the wordline work

MIPS Programming



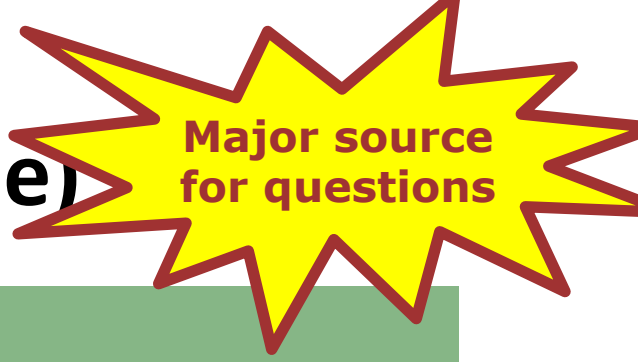
- **You should know**
  - How to write/read assembly programs for MIPS
  - Understand how function calls are made
  - How stack works and how registers are copied
  - How memory is organized (text, code, data segments)
  - Given the appendix of your textbook, be able to understand all MIPS instructions

What is the difference between round-down and rount-to-nearest, give an example where the rounding would result in different numbers.

- **We will NOT ask you**
  - Exceptions and exception handling
  - Floating point instructions

i.e. Explain what happens during an exception call. What is EPC and how is it updated.

Microarchitecture (Single Cycle)



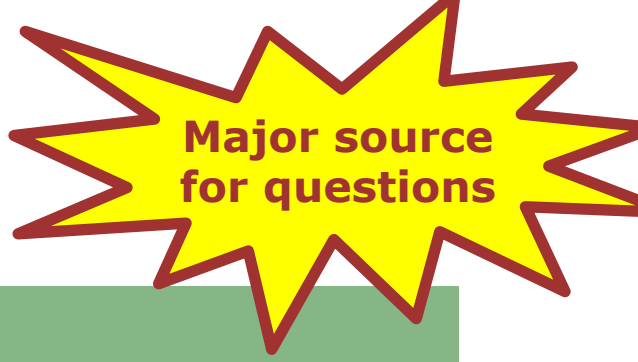
- **You should know**
  - How instructions are executed, and components of the processor
  - Calculate the performance of the processor and compare two processors with different parameters
  - How instructions are decoded and control signals for the processor are generated
  - Be able to interpret block diagrams of processors, find/add missing pieces

What is the difference between round-down and rount-to-nearest, give an example where the rounding would result in different numbers.

- **We will NOT ask you**
  - To know all instructions by heart, if questions involve details of an instruction, the table from Appendix B of your book will be provided
  - To draw an entire processor from scratch. If there are such questions, it will be to complete pieces of a missing diagram or find/fix errors.

i.e. What is the opcode for lw

Pipelined MIPS Architecture



- **You should know**
  - Pipelining basics, relationship between throughput and latency
  - Issues of pipelining, data/control hazards, mitigation mechanisms
  - Performance of pipelined processors

We have a single cycle processor that works at 1GHz. The idea is to have a 10-stage pipeline pipelining to improve performance. What is the theoretical max frequency? We use pipeline registers with a 20ps setup time, 0ps hold time and a 30 ps propagation/contamination delay. If everything else needed for pipelining can be done without ill effects, what is the max frequency we can achieve. What other effects do you expect would reduce this theoretical performance, name three different aspects, describe one of them.

- **We will NOT ask you**
  - To modify the block diagram to implement various hazard mitigation mechanisms like forwarding and branch prediction (you should still know what these mechanisms are, but not memorize the specific implementation details

i.e. Explain how the execute stage can be stalled and modify the block diagram of the processor to add the STALLD, STALLF and FLUSHE signals